SHX File Hacking

Revision 1

By **Eyesonly_d** at Howard Forums.

PRELIMINARIES

WARNING: This is modding to the extreme. Not for the light hearted. There is no sugar coating this, you screw up and your phone is toast. I am including everything I know and have discovered in this document, but keep in mind to tread cautiously, and be conservative in what you try to do.

Eyesonly_d is providing this information without liability. If you start hacking on your phone with this information you do it at your own risk, I am not responsible for dead phones as a result of this information.

This is also written for the advanced user with a considerable level of hex editing experience. If you don't know what words, bytes, offsets, bits, characters, endian-ness, etc are then just stop reading now. This file was created from a large pile of notes and the organization could be better, but this is as good as it gets for now.

Needed Utilities and Files

Hex Editor – Cannot stress this enough, a hex editor is your lifeline and your right hand in working with SHX files. Get a good one. The XVI32 that is recommended for seem editing is NOT enough. I use an older version of Hex Workshop, which is excellent. However I think Hex Workshop is no longer freeware.

RandomSHX – There are many SHX<=>BIN converters out there, this is the only one worth it's salt. Sorry to all the other authors out there. This is the only program that will faithfully recreate an SHX from bin files. The only thing it will not do is recalculating the header, which is what this guide is for.

Paper – Good for taking note and keeping track of things, also it is a good place to write down quick ideas and discoveries. There is a lot to learn in this domain of modding.

SHX files – Get lots and lots of these. Get 5 or 6 lang packs, get all the 9XR reflashes, get a few monster packs. When you are in doubt about how something should look, you should open up something similar and see how it looks. Or you can verify that a value stays the same for all E398 SHX files, or all E398 languages packs, etc. Also when going through these notes, open up the appropriate file in the hex editor and follow along, it will make you understanding MUCH easier.

A Little about SHX files

SHX files come in three types.

1] **Lang Pack –** Contains a set of languages and fonts for that language for a phone, allows a phone to quickly change available languages without changing the entire firmware. This does not change the user data, flash, or flex. Contain just Bootloader, and a Lang Pack. They are usually about 2MB in size

2] **Reflashes –** This allows a user to upgrade the Flash of the phone without touching the userdata (Flex). Contain Bootloader, Flash, DSP, Lang Pack, DRM, Signature. These around 33MB in size.

3] **Monster Pack** – Similar to the Reflash in file contents, but also includes a ROM dump of a flex. This is a complete packaging of firmware and will completely erase a users existing data when they flash it. Contains Bootloader, Flash, Flex, DSP, Lang Pack, DRM, Signature. These whoppers weigh in around 62MB in size.

An SHX file contains information about the ROM content of a phone. It is not in the same format as you would find it in your phone's ROM however. The format is roughly as follows.

First 8kb are the SHX "header"

- The first park of the header is always the words "P2K Superfile"
- Next are some arbitrary version numbers and a date that the file was created.
- Next is the clause "CSF Protected UNIX Generated Superfile" this is to identify the program typically used to create these SHX files.
- Next in the Hex range 0x3B0 to 0x600 is the information about the included memory areas of the SHX file. This is where most of your hacking will be done. Much more info on this later.
- Next you will find zeros until the actual data starts around 0x2000.
- This header is just plain chopped off the front of the file and dumped into BIN0 by RandomSHX.

Next is the "data area"

- Exact format is not known by me, but it is known by others who have successfully created the SHX to BIN programs.
- Different data sections seem to be differentiated by an S#####.
- Data is stored as a string representing the hex data to be written to the ROM, this is why SHX files are about twice as large as the E398 ROM (64MB vs 32MB). This is because each half a byte is being represented as a full byte character.

Using RandomSHX

RandomSHX can do twofold. First it can take the SHX file and convert it into raw bin format. Second it can take the raw bin format files plus a LST file and convert it back to an SHX file. The LST file stores the memory addresses where each bin file starts at. It is required for RandomSHX to put a SHX file together and it is generate by RandomSHX when you split an SHX apart. Here are some rules regarding LST files. Each rule takes precedence over all rules following it if not otherwise specified.

1] If LST files are from different phones they are not the same. Ex. An LST file from a V635 Lang Pack is not the same as a LST E398 Lang Pack.
2] Lang packs from the same phone will have the same LST.
3] Monster packs from the same phone will have the same LST
4] Reflashes from the same phone will have the same LST
5] V3 and E398 usually have the same LSTs and are an exception to rule 1.
6] LST files must have one (no more no less) line for every bin file.
7] An improper bin with wrong memory addresses will make the flasher put the data in the wrong place on the phone. This results in dead phone. Always a good idea to double check this file right before using RandomSHX to make sure all is kosher.

Bin Files – General

Bin files are the exact image of what the memory on the phone holds. For the most part they are random hex. But when looked at in a hex editor, bin files of similar type (lang pack, bootloader, flash, etc), will share patterns. Patterns like a large section of 0000 at the end, or always starting out with "1093 1074", or always have the phrase "Motorola App Signature" somewhere in the last third of the file. These patterns are how you can verify that the bin file you are looking at is really what you think it is. For the most part you will not edit a bin file. These are the result of a compiled machine code and any attempt to directly edit it would most likely result in an unstable crashing phone at best, and a dead phone at worst.

Also BIN files should not be mixed between phones. There are exceptions. It seems V3 and E398 can exchange LANG PACK, and FLEX.

The one thing important to all bin files is their length. This can be found by going to the end of the bin file and getting the offset of the last byte. This is NOT the end of the file offset, but the last byte, meaning one before the end of the file.

The next important thing is the checksum. A checksum is where you take all the bytes and add them all together and you keep the last 4 digits. Later on if you flash the phone MFF will perform the same check. If it's checksum doesn't match what you tell it that it should be, MFF will report that the code on the chip is corrupt.

It seems that the different bin files record their checksums differently and I don't know the algorithms. However a common thing I noticed is that you need to convert the word's endian scheme before calculating a checksum.

Below is a summery of all the BIN files. In each summery is a section about its header, this refers to an offset in the BIN0 file. The BIN0 file is the same as the first 8kb of the SHX file and stores all the information about the bin files included in the SHX package.

In these header descriptions the hex numbers you will need to change are shown as "####", if the number is shown as a constant then usually you will not need to change this number, and usually by changing it the flash file will fail. This isn't always the case, see the attached notes for each offset for more information.

BOOTLOADER BIN

This section holds the bootloader of the phone. MFF usually does not flash a bootloader, but RadioComm will. The bootloader is included will all types of firmware files.

Header format stuff:

```
0x3B0] 0800 0000 0000 FD03 F7FF FE03 03FE F800 B172 19E9 000C 0203 0100
       (1)            (2)  (3)  (4)  (5)                 (6)       (7)
       07E0 ####
       (8)  (9)
```

(1) – Always the same for the first three words
(2) – This is the starting address in ROM in backwards endian format.
(3) – This is the offset of the last byte in the bin in backwards endian format.
(4) – This is the ending address in ROM in backwards endian format.
(5) – This is the address in ROM
(6) – These bits are found a lot in the header, may be related to hardware.
(7) – This is sometimes 0200. Not sure why, I couldn't find any correlations
(8) – This is the version of the bootloader, some flashing softwares will not flash a software unless it is the same bootloader or higher.
(9) – Checksum, unknown algorithm (8DCC for 98R and 99R reflash files)

FLASH BIN

This section holds the Flash, the kernel OS of the phone. There is lots to look at in its header part.

Header format stuff:

```
0x3EE] 0810 C700 0810 #### #### 0000 00B1 000C 0206 FFFF 0E20 E88F 0200 C800
       (1)  (2)  (3)  (4)  (5)    (6)         (7)       (8)  (9)    (10)
       0810 #### BE10 #### #### 0000 00B1 010C 0206 FFFF 0E20 ####
       (11) (12) (13) (14) (15)  (16)         (17)      (18) (19)
```

(1) – This is the FLASH's starting address in ROM in backwards endian.
(2) – This a small length, its always the same
(3) – This is the FLASH's ending address in ROM in backwards endian, these three words above seem to suggest that the first section of the flash is separate from the rest. Why this is the case is unknown.
(4) - This is the first word in the flash bin file.
(5) - This is the second word in the flash bin file.
(6) - These two words keep showing up 0000 001B.
(7) - These three words also keep showing up 000C 0206 FFFF, may be hardware related or specify options on how to flash the bin.
(8) - 0E20 – You will recognize this from a part in the file name, this specifies what hardware this flash is for. (0E20 is for E398, 0E40 is V3)
(9) - E88F – A checksum of sorts, but it is always the same value no matter what.
(10) - This looks like a repeat of the first three words except the first word is 0200 and the second is C800 instead of C700.
(11) – This is first two bytes the flashes starting address in ROM in backwards endian. (10080000)
(12) – This is the offset of the last byte in the bin in backwards endian. So if the last byte is at address 0x01F0BC42, then this value is "42BC"
(13) – This is the first two bytes of the flashes ending address in ROM in backwards endian. (10BE0000)
(14) – This is the first word in the flash bin file. (same as 4)
(15) – This is the second word in the flash bin file (same as 5)
(16) – These two words keep showing up 0000 001B, they may be related to how the BIN file is to be flashed.
(17) – These three words also keep showing up.
(18) – 0E20 – Specifies the hardware this flash is for.
(19) – Checksum, algorithm unknown.

When an SHX file is flashed, one of the steps is "Erasing Flash memory" I believe part of this header specifies how that erasing occurs.

FLEX BIN

This section holds the Flex, the flex contains the filesystem and the seems. If you browse through this file you can see filenames and structure. You can even find the seems. There seems to be two copies of the seems. One is near the beginning, the other is near the end. Files are dispersed throughout. Empty space is denoted by large areas of FFFF. The flex bin is always 14.6MB, any space not used is the free space for the phone's user. In general flexes can be mixed and matched with little worry, some are better than others, and some don't enable all functionality that a flash provides.

Header format stuff:

```
0x426] 1411 FFFF FD11 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF #### 0800
       (1)       (2)  (3)                                     (4)  (5)
```

(1) – 1411 – This is the first two bytes of the starting address in ROM inbackwards endian.
(2) – FD11 – This is the first two bytes of the ending address in ROM in backwards ending.
(3) – 8 words of FFFF
(4) – Checksum, unknown algorithm
(5) – Always 0800

DSP BIN

This section holds the DSP code. You can check this on your phone in the "Other Information" menu. Many E398 firmwares have the same DSP, and you can generally mix and match them as you please.

Header format stuff:

```
0x442] 0410 #### 0710 #### #### 0000 00B1 010C 0206 FFFF 0E20 ####
       (1)  (2)  (3)  (4)  (5)  (6)       (7)            (8)  (9)
```

(1) - This is the first two bytes of the starting address in backwards endian.
(2) - This is the offset of the last byte in backwards endian. So if the last byte is at address 0x01F0BC42, then this value is "42BC"
(3) - This is the first two bytes of the ending address in backwards endian.
(4) – This is the first word of the bin file.
(5) – This is the second word of the bin file.
(6) – Sequence that shows up a lot 0000 001B
(7) – These bytes show up a lot too 010C 0206 FFFF
(8) – 0E20 this is just like the file name for the SHX file, this specifies what hardware this works on.
(9) - Checksum, unknown algorithm.

Lang Pack BIN

This section holds the languages and the fonts for those languages. They range in size from 700kb (lang pack 0003) to 1.2MB (lang pack 0015) from what I have seen.

Header format stuff:

```
0x45E] F410 #### FF10 10F4 0010 0000 00B1 010C 0206 FFFF 0E20 ####
       (1)  (2)  (3)  (4)       (5)       (6)            (7)  (8)
```

(1) – This is the first two bytes of the starting address in backwards endian.
(2) - This is the offset of the last byte in backwards endian. So if the last byte is at address 0x01F0BC42, then this value is “42BC”
(3) This is the first two bytes of the ending address in backwards endian. This can change between short language packs (like 0003) and longer language packs (like 0015), when in doubt, look at an official language pack of that version and see what value is here.
(4) Address in forwards endian
(5) Sequence that shows up a lot 0000 001B
(6) These bytes show up a lot too 010C 0206 FFFF
(7) 0E20 this is just like the file name for the shx file, this specifies what hardware this works on.
(8) Checksum, unknown algorithm.

DRM BIN

Not sure what this is used for. At the beginning of the BIN is always the word DRM so I just called it that. Most E398 firmwares list this as DRM0001 and are identical except for one part where a text string of the flash is. Since the flash differs in different SHX files, then this text string can differ. Also DRM0003 exists and is much smaller (about 500kb smaller). It was found with 96R_A. In general if you found a certain DRM version with a flash, then you need to keep it together because the text string specifies it to that flash.

Header format stuff:

```
0x592] D010 #### EF10 0000 0000 0000 0000 10D0 001C 0000 001B ####
       (1)  (2)  (3)  (4)                 (5)       (6)       (7)
```

(1) – This is the first two bytes of the starting address in backwards endian.
(2) – This is the offset of the last byte in backwards endian, So if the last byte is at address 0x01F0BC42, then this value is “42BC”
(3) – This is the first two bytes of the ending address in backwards endian.
(4) – Four words of 0000
(5) – Address in forward endian format
(6) – Sequence that shows up a lot 0000 001B
(7) – Checksum, unknown algorithm.

Signature BIN

Not sure on what this is used for. I just started calling it “Signature” and it stuck.
Can be verified by the test string “Motorola” being found all over the place in the bin file.
Always ends in a long bunch of FFFF.
This is different for every Flash, so it should always be paired with its correct flash.

Header format stuff:

```
0x5E6] FE11 FF07 FE11 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF #### 0800
       (1)  (2)  (3)  (4)                                     (5)  (6)
```

(1) - FE11 – This is the first two bytes of the starting address in backwards endian.
(2) – FF07 – This is the offset of the last byte in backwards endian, means to 07FF.
(3) – FE11 – This is the first two bytes of the ending address, in this case it is the same as the starting address because the signature is so small.
(4) – 8 words of FFFF
(5) – Checksum – This is found by flipping the endian of the file (good hex editors can do this) and then generate a 16 bit checksum. (0B86 for 98R) (0183 for 99R)
(6) – This is the length of the BIN.

Lang Pack Organization

LST – E398 example

```
This file is created by Random's SHX Toolkit
File: 1
Addr: 03FD0000
File: 2
Addr: 10F40000
```

BIN0 – HEADER – 8kb always
BIN1 – Bootloader – Not used by MFF, always identical among E398 lang packs. 128KB always.
BIN2 – LANG PACK – Around 1MB +/- 300KB

Reflash Organization

LST – E398 example

```
This file is created by Random's SHX Toolkit
File: 1
Addr: 03FD0000
File: 2
Addr: 10080000
File: 3
Addr: 10040000
File: 4
Addr: 10F40000
File: 5
Addr: 10D00000
File: 6
Addr: 11FE0000
```

BIN0 – HEADER – 8kb always
BIN1 – Bootloader – Not used by MFF, always identical among E398 lang packs. 128KB always.
BIN2 – FLASH – 10MB to 11MB
BIN3 – DSP – around 200KB
BIN4 – LANG PACK – Around 1MB +/- 300KB
BIN5 – DRM – around 2MB (must match flash version)
BIN6 – SIGNATURE – 2KB (must match flash version)

Monster Pack Organization

```
This file is created by Random's SHX Toolkit
File: 1
Addr: 03FD0000
File: 2
Addr: 10080000
File: 3
Addr: 11140000
File: 4
Addr: 10040000
File: 5
Addr: 10F40000
File: 6
Addr: 10D00000
File: 7
Addr: 11FE0000
```

BIN0 – HEADER – 8kb always
BIN1 – Bootloader – Not used by MFF, always identical among E398 lang packs. 128KB always.
BIN2 – FLASH – 10MB to 11MB
BIN3 – FLEX – 14.6MB always
BIN4 – DSP – around 200KB
BIN5 – LANG PACK – Around 1MB +/- 300KB
BIN6 – DRM – around 2MB (must match flash version)
BIN7 – SIGNATURE – 2KB (must match flash version)

CHECKSUM FINDING

If you are unable to determine a checksum by looking a original headers, or calculating it on your own, there is another way. First create the SHX file with the wrong checksum values. Now use MFF and attempt to flash this new SHX file to the phone. The file will flash, but at the end MFF will report a checksum error. Do not close the program or pull the cable! So to the directory where the SHX file lies and open up the .log file found there. Inside it will report the checksum errors for each numbered bin. For each bin with a bad checksum there will be a line saying “File: XXXX Phone: XXXX” where XXXX are hex values. Write all these down. Now go back to your header file to the location of the bad checksum values. The values should read the same value reported next to the “File: XXXX” part in the error log. Verify this. Now change this value to the “Phone: XXXX” value. Recompile the SHX with the new header and everything should work.

PUTTING IT ALL TOGETHER

Once you fix your header to the correct values you can recompile the SHX file with Random SHX. After this you will have a working SHX file. May I make the following recommendations.

- Use file names with the same format as other file of the same type. Don't just make the SHX file "MyGermanSHX.shx." It will still work, but it is much more useful to include the official file name. This official file name includes information about what phone it works on, what flash it came from, the number of the lang pack, etc.
- Put a phrase at the end so people will know this is an artificial firmware, not and official one. I always put "EYEZHACKED" at the end of my file names. This also will help you keep track of what is official and what is artificial. Also this helps give you credit for your work, and lets everyone know who to blame if it doesn't.

E398 ROM DUMPING

In addition to taking apart and reconstructing SHX files, one can also incorporate a ROM dump from a phone. A ROM dump is where an exact copy of a phones ROM memory is created and "dumped" into a file. The only program able to do this on the E398 is RAMLDR by Vilko (maker of P2kMan). Here are some instructions on how to use it.

1] First put the phone into flash mode
2] Start up RAMLDR
3] Press "Send RAMLDR" and select the ldr.bin file.
4] Now we set the memory range we want to dump.

For the E398:
BOOTLOADER: 03FD0000 03FF0000
FLASH: 10080000 to 10D00000.
FLEX: 11140000 to 11FE0000
DSP: 10040000 to 10080000
LANGPACK: 10F40000 to 11140000
DRM: 10D00000 to 10F00000
SIGNATURE: 11FE0000 to 11FE0800

Now click "Save Mem" if all goes well the program should start churnging along with DUMP mesages.

When it is finished there will be a file named as the start address in the RAMLDR folder.

When you are done be sure to hit the reset button to bring your phone back to normal state. While the phone is reseting .. pull the cable out so RAMLDR does not force it into flash mode again.

Now the Bootloader, FLEX, and SIGNATURE will just need to be renamed to their appropriate BIN name.

The Flash dump will have a long section of FFFF at the end. You can safely trim off these FFFF. The resulting file should be around 10MB-11MB.

The DSP will also have FFFF at the end, and also there is a charectoristic hex patturn at the end of all DSP BINs, look for it and trim the dump to make it match in length.

The Langpack will end in FFFF at the end, trim it like the others.

The DRM will end in FFFF at the end, trim it like the others.

Now rename all the files for their appropriate bin number, include a LST file, reconfigure the header with file length and checksum info, and compile it with RandomSHX and you will have a monster file backup of your exact phone. Lang packs and reflashes can also be made in similar fashion as long as you adhere to the formats specified in this document.

WHAT YOU CAN AND CANNOT DO WITH THIS KIND OF MODDING

CAN:

Mix and match language packs, flashes, and flexes of a certain phone to create new monster packs.

Extract language pack bin out of a reflash or monster pack and create a new lang pack.

Create new reflashes, monster packs, and lang packs based on the data on a phone ROM.

Create a complete single flash phone backup.

CANNOT:

You can't mix and match part of a single BIN. You can't combine the Chinese part of one language pack into the German part of another.

You can't extract features out of a flash and put it into another, IE you can't take video recording out of a V3 or V620 flash and put it into a E398 flash.

You cannot haphazardly mix BIN files from different phones. The files must fit the same size and format requirements, and even if they do the hardware on the other phone will probably be too different for the phone to even boot up after the flash.

Take a BIN file and convert it to an HS file directly. You can flash the flex bin to the phone, then use a program like Motokit to create a flex from the files on the phone. (This is an indirect way to accomplish the same thing)