The technical guide to MTK Phones

By
apraxisz
PhiKore.com

Edition 1

Preface

As someone with some experience in the embedded development field I will try to explain the procedure to successfully hook an MTK phone up to a computer. I've written this to be as thorough and logical as possible. My goal was to provide you with THE one e-book that you need in order to start working with MTK based phones. I hope you find this document useful. Let's begin.

Introduction

What are we dealing with?

My preferred way of looking at MTK based cell phones is that of looking at small ARM based embedded platforms. The MTK baseband chips have 33-66 MHz ARM cores. They come with 4-8 megabytes of SDRAM, and either a NOR or NAND flash for storage.

☺	Q: Is it possible, in theory to run Android on these phones? A: No. Android has much higher minimum requirements.

Our goal when interfacing with these phones is to read/write the contents of the flash chip, so we will focus on that for a bit.

The flash chip contains 2 things: the firmware and a file system section. The firmware of course is the program (OS) that works the phone. It handles user interface, communicates with all the attached devices (microSD card, camera, BT module, GSM module, etc..) it's the central piece of program.

Firmware

MediaTek sells a reference firmware platform to its licensees which they all use (after varying levels of modification). What this means is that all MediaTek firmwares are based on the same code base, and thus all MTK phones have similar, if not identical basic features. For example: basic UI code, such as language handling, input handling, and resource handling code will be the same. This is why a single MTK Firmware Editor can open almost all MTK firmwares, and successfully edit the resources contained within.

This reference platform comes with no support for external executables. All functionality is linked into the firmware at compile time, so adding new features or applications afterwards is infeasible, and without months of reverse engineering, by skilled developers, impossible. There are no such projects so at the time of writing there is no way to run outside code on the MTK platform (unless the firmware of interest has Java support, or NES emu compiled in).

☺	Q: Can I add new apps to the phone? A: No. (Unless your firmware has Java support or NES emu) Q: Can I create my own apps that run natively on the phone's CPU? A: No.

Usually release firmwares come compiled with a specific set of drivers for the target hardware configuration. Different model phones will not have the same type of LCD display, FM tuner chip, BT chip, camera(s) etc.. But there's no guarantee that same model phones, manufactured on different dates do either. Indeed same model phones manufactured just a few weeks (or days) apart can have different hardware. It is because of this that 5 different firmware versions of Model X could have been release not because of firmware improvements/bugfixes, but because the hardware in the phone has changed and new drivers were required.

What this also means is that most of the time trying to “upgrade” the firmware on these phones will result in bricking, or the loss or malfunction of an attached device (display, camera(s), BT, etc.)

For example: White screen, black screen, no screen, after flashing, but hearing the phone play the startup chime is because the phone’s display is either different, or connected differently in hardware, and the driver in the firmware we’ve flashed isn’t able to handle it. Similarly, loss of camera support, or malfunctioning keys are all symptoms of this.

A number of firmwares MAY have several drivers compiled into them, with the right one being selectable from the Engineer Menu, so in some cases that may work. Also a model’s hardware configuration may be stagnant, because of an overabundance of the parts used in the original design, in this case upgrading would be feasible, and likely worth it. Sadly however, this is not the case most of the time. It is because of this, and other factors that I’ll talk about later, that it is HIGHLY recommended to **ALWAYS BACK UP** the **FULL flash image**, that was originally on the phone. 5-10 minutes of annoyance, can save you hours/days of frustration.

☺	Q: Can I use firmwares from other MTK phones? They have features I want. A: 90% chance the phone won’t even start. 100% it won’t work entirely right. Q: Can I upgrade the firmware version on my phone? A: You can try, but ALWAYS BACK UP the original flash content, as it’s unlikely that you’ll get the desired outcome.

Identifying firmwares by version

MTK firmwares have a so called, version code, that when entered will output the firmware version of the phone. You can find a set of these in Appendix A, but the most common ones are *#8375#, and *#4853*#
The phone should immediately display the version screen once you enter the correct code. You should not need to press anything else. If it does not, then you’re not trying the right code for your phone. It is also likely, that your phone has a code that isn’t in the Appendix, as one of the first things some firmware developers seem to do is change these to a random value. When entering these codes you will get a version string (ex.: A6V2.2.2) and a Build Time (ex.: 2007/12/04 18:31)
Now you might think that checking just one of these would be enough to identify the firmware, and you would be wrong. The two firmwares below are different, and the hardware of the two phones doesn’t match. So flashing one of these to a phone that had the other one on it originally, will produce a white screen, and unusable phone.

- A6V2.2.2 - [BUILD TIME: 2007/12/04 18:31]
- A6V2.2.2 - [BUILD TIME: 2007/12/18 14:46]

This can also happen the other way around with Build Times being identical and version strings being different. It is my experience that comparing both string and build time is sufficient to discern firmware versions, but I usually read back the first 10kilobytes of the firmware from the phone and compare it with the version I have on file, just to make sure.

File system

The file system section comes after the firmware on the flash chip. Different flashing solutions call it differently including FAT, FFS, NVRAM, System/User FS, etc..

This section contains several actual file systems (FAT12, FAT32) on top of a proprietary structured data store layer. I know of no software that can handle this top layer, so access to the file systems within is only possible through the phone itself for the time being. These file systems contain all the data used by the firmware from system data like device driver settings, to user data like the phonebook, messages, or data account settings. When people refer to formatting a phone, they're referring to these file systems. Formatting as with formatting file systems on regular PC hard disks, or flash drives erases all data stored in these file systems. Upon starting the phone after a format it will recreate all of its needed files inside these file systems, and fill them with hardcoded defaults. This includes all calibration data, and device driver settings. Contrary to popular belief formatting these file systems without backing up is anything but safe. As hardcoded defaults can differ from factory set values, so there could be several things that you're influencing by letting the phone use the hardcoded defaults. It is best to back up the FAT of each phone that passes through your hands, as they're only a small portion of the flash content so it takes about 15-30 seconds, and once again: You may be saving yourself from lots of frustration in the long run. What you should keep in mind about the file system is that IMEI, and personal data are also stored here, so you may want to think twice about publishing your firmwares with FAT included, because of the privacy issue. Also if the people who end up using your firmware are ignorant to these facts, they could end up making several phones that use the same IMEI (Yours!). Which depending on carrier, may result in banning the IMEI from the network, instantly rendering all phones that use it, inoperable, or another case, where a phone with said IMEI is reported stolen.

Supposing you have 10 phones with the same firmware, you download the firmware from Phone #1 modify it with an editor, save a Full Image then flash said image onto Phones #2-#10. You now have 10 phones with the SAME IMEI. What you should've done is either back up the file systems from all the phones and write the originals back after the full flash, or if possible, patch-flashed your phones. (This means flashing only the parts you wish to modify and leaving the rest of the flash chip intact, this isn't always possible.)

Another thing to keep in mind is that there could be factory protections that use the file system. For example the PSN protection: in this protection form, a serial code generated from a unique hardware ID is stored in a file on the file system. If you flash a full image from one of these phones onto another, even if they're the same version, the target phone will start up with "Register Mercury" And will STOP FUNCTIONING after a few weeks. If you haven't backed up the file system of such a phone you essentially have a phone that needs a file system reformat every few weeks to start up, or in other words, a phone that can't be sold. Few repair boxes can fix PSN, and they're expensive. So once again: The thing to keep in mind is to **ALWAYS HAVE A BACKUP.** Best is to have full backups, but at the very least, if you already have a full backup for the specific firmware version on the phone, you should still take that 1 minute to back up the file systems from all the phones. You just may be thanking yourself later.

☺ Q: What should I back up?
A: At least 1 full backup per fw version, and the file systems section for each phone.

Interface

There are usually at least 2 ways in which you can interface with an MTK phone. With the supplied data cable: through USB, through a 3.3v serial UART, and through BT.
The grid below illustrates what you can do with each interface.

	DataSuite	Modem	Flashing	Mass Storage / WebCam / PictBridge / USB virtual UART /etc..
USB Data cable	X	X		X
Serial UART	X	X	X	
BT/IrDA	X	X		

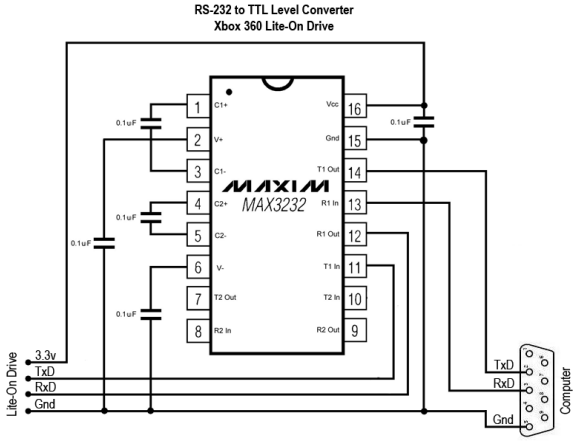
As you can see, you can't access the flash chip with the supplied USB Data Cable. You will need to buy an interface for the Serial UART of the phone if you wish to access it. This is unfortunate, but there's no way around it.

But wait, you say. Your phone has a COM Port in the USB Menu, so surely you could use that. You could not. The usb virtual UART cannot be active at the time it would be needed to boot and flash the phone, as it is only active while the firmware's running.

The RS/EIA232 Serial UART (or commonly just: serial port) is a very common interface supported by almost every computing platform in the world. Older PCs have COM Ports for example. Those would be an example. Most microcontrollers have either software or hardware based UARTs (universal asynchronous receiver/transmitter). Serial ports require at the minimum 3 wires to work. One Tx(that the device transmits on) one Rx(that the device receives on) and the ground. There are however differences in electrical specifications, and supported data transfer rates. The MTK platform is a 3.3v based system. That means that that most everything in the phone is running off 3.3volts that is generated by a voltage regulator connected to the 3.7v battery. The maximum voltage for logical HIGH states is 3.3volts. The serial port works at 3.3volts as well. On the other hard, a PC's serial port works at 12volts. You do not wish to connect a 3.3volt system to a 12 volt system directly ☺

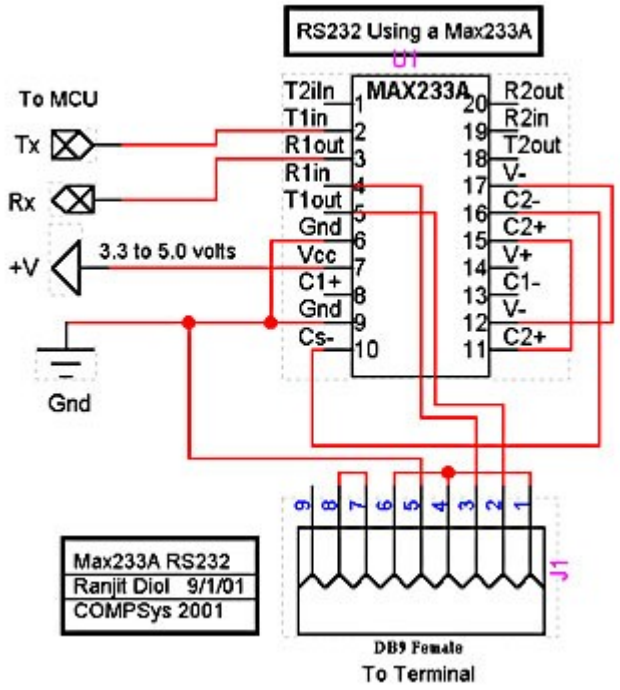
Direct COM Interface (slow)

This is where level shifters come into play: one example of an RS232 level shifter would be the MAX3232.



If you build the above schematic, and supply it 3-5 volts (Phone's battery works fine) you would be ready to hook up to your MTK phone through your PC's Com port. Great! If you have one of these chips lying around, this would be your cheapest and quickest solution. You could even order them as FREE samples from Maxim/Dallas, or get clones of them (ex.: HIN3232) cheap.

Another option to further simplify the circuit above would be the MAX3233, which has integrated capacitors.



Be sure to use 3233, and not 233. 233 works on 5v, and the phone battery only supplies around 4.2 when fully charged. (In practice 233 would most probably work just as well, but if you have a choice to get either one, get 3233.)

However PC serial ports regardless of what their drivers' report are limited to 115200 bits per second.

Let's do the math:

(16 megabytes) / (115 200 bits) = 1 165.08444

1165 seconds = 19.4166667 minutes

20 minutes to read a common 16megabyte firmware chip. Not the fastest solution ever invented. 32 megabyte chips are gaining ground in the MTK platform, but I've already seen 64 and 128 megabyte NOR chips as well.

(64 megabytes) / (115 200 bits) = 4 660.33778

4660 seconds = 77.6666667 minutes

If you're patient, this solution would indeed work for you. But it's far from ideal, and is not generally used except by hobbyists who are just looking to flash their own phones once in a while. If you feel that you fit in this category, go right on ahead.
See these URL for free samples of the mentioned chips:
http://www.maxim-ic.com/quick_view2.cfm/qv_pk/1068/t/or (MAX3232CPE, MAX3232CPE+, MAX3232EPE, MAX3232EPE+) external capacitors
http://www.maxim-ic.com/quick_view2.cfm/qv_pk/2008/t/or (MAX3233ECPP+G36, MAX3233EEPP+G36, MAX3233ECPP, MAX3233EEPP) no external capacitors needed

USB Serial Interface

The serial ports on MTK phones support up to 460800 bits per second. But how can you take advantage of that with a PC? Through a USB Serial cable. Unlike regular PC serial ports USB Serial chips are capable of speeds of up to 921600 bits per second. The MTK phone will initiate communication at this speed, but it won't be stable, only at 460800 does it work reliably. Let's check the math again.

(16 megabytes) / (460 800 bits) = 291.271111

291 seconds = 4.85 minutes

Now that's a lot more promising ☺ In practice you can get close to 5 minutes during read-backs. Flashing is usually somewhat slower.

That said, you shouldn't go out and buy just any USB Serial cable, as the ones that come with a DB9 socket, and sold as USB Serial Ports were made to be used as replacements of regular PC serial ports. So you would probably have the same problems with them as with regular serial ports. They might have on-chip level shifting, that you can't even disable, or have extra chips that you would need to remove. What you should look for are flashing cables sold for cell phones. There are several available online for an assortment of models, most of them can be modified for the purpose. The procedure differs based on what kind of serial port the phone has that the cable was made for, and what kind of usb-serial converter chip is used, and whether the cable has any plus features, or the phone any additional steps required to initiate communication on the cable.

For example I used a PL2303 based Nokia DKU-5 clone.
The PL2303 has a special pin that sets the voltage at which the serial port should work. This had to be connected to a 3.3volt regulator that I had to add externally as the PCB of the cable was in a "poured" connector, and couldn't be accessed without force, and possible damage. PL2303 based cables don't necessarily have this pin exposed. They could be connected to a 3.3volt regulator on-board, in which case no modification would be needed to use it.

Another time with a Samsung cable, I had to remove an extra resistor that was required for Samsung phone interfacing, but was interfering with MTK phone communication.

Probably the simplest solution would be to buy a pre-made cable for your model. That way you don't need to mess around with the interface, or hooking up to the right pinout by yourself.
Search at ipmart.com
Search for <your model> flash cable or <your model> unlocking cable
Try other sources such as u-cables.com
If you can find a pre-made cable for your model, great! If not, you should get one that's for an MTK phone. For example search for "China15 unlocking" (without quotes) on ipmart.com

These are FTDI chip based cables. FTDI chips are the chips of choice for usb serial interfacing, so they should be ideal for this purpose, and since you're buying a cable that was intended for another MTK phone you shouldn't need to modify anything, except the connector.

You will also need to buy a connector for your phone. Search for "china connector" (without quotes) on ipmart.com, and find the right one. (You might want to order a few spares as well)

Physical connection

If you have a pre-made cable, you can skip over this.

If you're going the USB route, you will need to remove the original phone connector from your cable at this point. You can take it apart and desolder or just cut it off. You need to find out which wire is what. If you bought the suggested China15 unlocking cable (CECT K86) You can go by the pinout here: http://pinouts.ru/CellularPhones-A-N/chinese_cect_K86_pinout.shtml
I'm going to assume that you now have your GND, RX, and TX wires, separated and stripped, regardless of which interface you're using.
You now need to open up your new connector, and search for the pinout of your model. Consider these two ways of doing this. Online search (which is not always reliable because of hardware variations and uncertainty regarding model number) and the Multimeter method.

The Multimeter method

1. Plug your new connector into your phone, with solder pads/pins exposed
2. Put Multimeter into fault-check. (or if unavailable, then it's highest resistance range)
3. Keep negative Multimeter probe on negative battery connector in phone.
4. Touch each pin of the connector in sequence with the other probe until you hear a beep (or see 0.000 resistance)
5. Note this connector pin as GNDPin
6. Now install battery and switch on Phone
7. Put Multimeter into DC voltage measurement
8. Keep negative Multimeter probe on GNDPin
9. Touch other pins of the connector in sequence and note pins that have voltage between 2.6 and 2.9 volts. (be careful not to short pins ☺)
10. You should end up with a list of 2 or 3 pins. RX and TX will be among these.
11. Try all combinations of RX and TX with these pins you found.

I've found several hundred pinouts with the method above. It works 99% of the time, and takes about 2 minutes once you're in practice.

So you now have your RX, TX, GND wires on your interface end, and your pinout or set of pinouts to try for the phone. It's time to dive into the flashing in practice.

But first…

Flashing

Flashing theory

After physical connection has been established, flashing the phone happens the following way:
Phone is turned off.
Boot process is initiated on the PC in the flashing software. This continuously starts sending a certain byte out on the serial port.
The phone's power button is pressed, phone reads it's com port, and if it detects this byte it goes into remote boot mode, otherwise starts the firmware. The piece of software that does this is an internal bootloader that is inaccessible and protected. Meaning, that you will never lose the ability to flash your phone, as long as the hardware is intact, regardless of how much you mess it up while flashing. You can't corrupt the bootloader, so you will be able to connect to your phone even if the flash chip is completely empty.
Once the phone has entered remote boot mode, the flashing software sends it a piece of code called a Download Agent, or DA. The bootloader loads the DA up into ram and starts executing it. This piece of code is actually what's doing all the hard work related to flashing: Detecting the onboard flash chip, reading and writing content and communicating on the serial port. The flashing software on the PC side merely sends commands to the DA from this point on.

DA Bugs:
It's worth mentioning that the DA, like all other pieces of software can have bugs. As the DA is a closed source piece of proprietary software, these are also unlikely to be fixed. One such persistent bug is that certain 32meg+ flash chips cannot be patch flashed.
What's patch flashing you ask? Patch flashing is writing only a specific section into the flash chip, say 414 bytes starting at 0x223421, instead of everything from 0x0 to <CHIP_SIZE>

Full-flashing theory

Now we're getting somewhere. Full-flashing means writing/rewriting all content on the flash chip, from an external source. For example, for a 32 meg flash chip, we have a 32meg .BIN raw flash image on our PC; all data in the file will be written to the flash chip.

MediaTek has decided to make this hard for everyone, by implementing what is called a Security Zone. The security zone is a data section inside the firmware image that is cryptographically tested by the firmware (or bootloader), and failing this test results in the phone not turning on. So a full flash procedure with the freely available MTK FlashTool most of the time results in a bricked phone, unless the individual developer for the phone has decided to disable Security Zone protection.

Security zone details:

Successful full flashing requires an “open” security zone. During full-flashing the security zone is encrypted (locked) with a hardware unique code, so from that point on the firmware image works only on that specific phone.
Another level of annoyance is added because when we read a full flash image from the phone, what we get is the locked security zone. Flashing this same image back onto the SAME phone also produces a brick, as the already locked zone is locked once again, the test fails and the phone doesn’t start up.

So how do we go about full flashing?

There are two methods to go about this depending on your application.
If you wish to full flash a phone with it’s own firmware (or a modified version of it’s own software, for example language change, or resource edit) you can do the following process:
(Here we’re using MTK FlashTool, for usage instructions see end of book)

- Read a full backup, name this original.bin, back this up in a safe place
- Full flash this backup to the phone. (at this point the phone is dead.)
- Read another full backup, name this broken.bin

Now you have two files that should be logically identical if you haven’t read the above section, but since you did, you know that original.bin will have the original locked Security Zone, while broken.bin will have a Security Zone that has been locked once again (broken). This has effectively made the system reveal to you where the security zone is. You just need to compare the two files for example with the windows command line tool fc.

What you can do now is patch-flash the correctly locked Security Zone from original.bin onto the phone. Since you’re not full flashing this time, the locking will not be triggered again, so you can replace the broken zone with the correct one using this procedure.

- Run fc /b original.bin broken.bin >a.txt
- Open the newly generated a.txt, you will see:

Comparing original.bin broken.bin
002DB6C0: CA 1F
<more lines like the above>
002DB733: 5B 00 (bytes after address irrelevant)

- In the above case, the range would be 0x2DB6C0 to 0x2DB733, note down yours

Now you need to open original.bin in a hex editor, and copy the range into a new file. The way to do this will vary depending on hex editor. Simple way to do it in XVI32:

- Ctrl+O Open file, select original.bin
- Ctrl+G, select hexadecimal, select absolute, $2DB6C0 (your range start)
- Ctrl+B (block selection start, nothing will happen)
- Ctrl+G, $2DB733 (your range end)
- Ctrl+B (block selection end, a block will be highlighted red)
- Ctrl+C (copy block to clipboard)
- Ctrl+N (create new file)
- Ctrl+V (paste clipboard)
- Ctrl+S (save) enter a filename, save it somewhere, let’s say SECAR with no extension

So now you have a file called SECAR, containing the correct Security Zone for you phone, you need to create a scat.txt for MTK FlashTool patch-flashing.
Open up notepad and type SECAR 0x2DB6C0 (your range start)
Then save this file as scat.txt in the same directory where the SECAR file is.

Now you can patch-flash the security area (see end of book for MTK FlashTool usage guide) and ta-dah! The phone comes back to life.
I know what you're thinking, this is foobar complicated, and indeed it is. This is THE one way to fullflash without a box, save for using cracked box software. The question arises, do you NEED full flashing for resource and language modifications? Not necessarily.
However, please recall what I've written about DA Bugs. It would be safe to say that if your phone has a 16 meg flash chip, you can patch-flash it all you want. However, the problem is with the 32meg chips that can't be patch flashed. If you take into account that we also needed patch flashing for the above boxless full flashing procedure the problem becomes evident. At the very least we need box functionality to reliably flash phones with 32 meg or larger chips.

We do not condone the use of cracked software.

Boxes

Flashing boxes are capable of opening the Security Zone before a full flash. No, let me rephrase that: The software that comes with flashing boxes is capable of opening the Security Zone before a full flash, allowing the full flashing of phones; this is actually one of the many things that warrant their existence. Boxes are essentially hardware keys that protect the software that was coded for them, combined with USB Serial adapters. Some also include dynamic pin configuration and automatic pinout finding circuitry. They can cost several hundred USD, but come with support, cables for most MTK phones and flashing software. Some also provide firmware editing software, either in the basic package or as an addon for a fee. Qualities of these vary from box to box. I will not be singling out any boxes in this e-book, as I only have experience with a few. And those weren't the most popular ones.

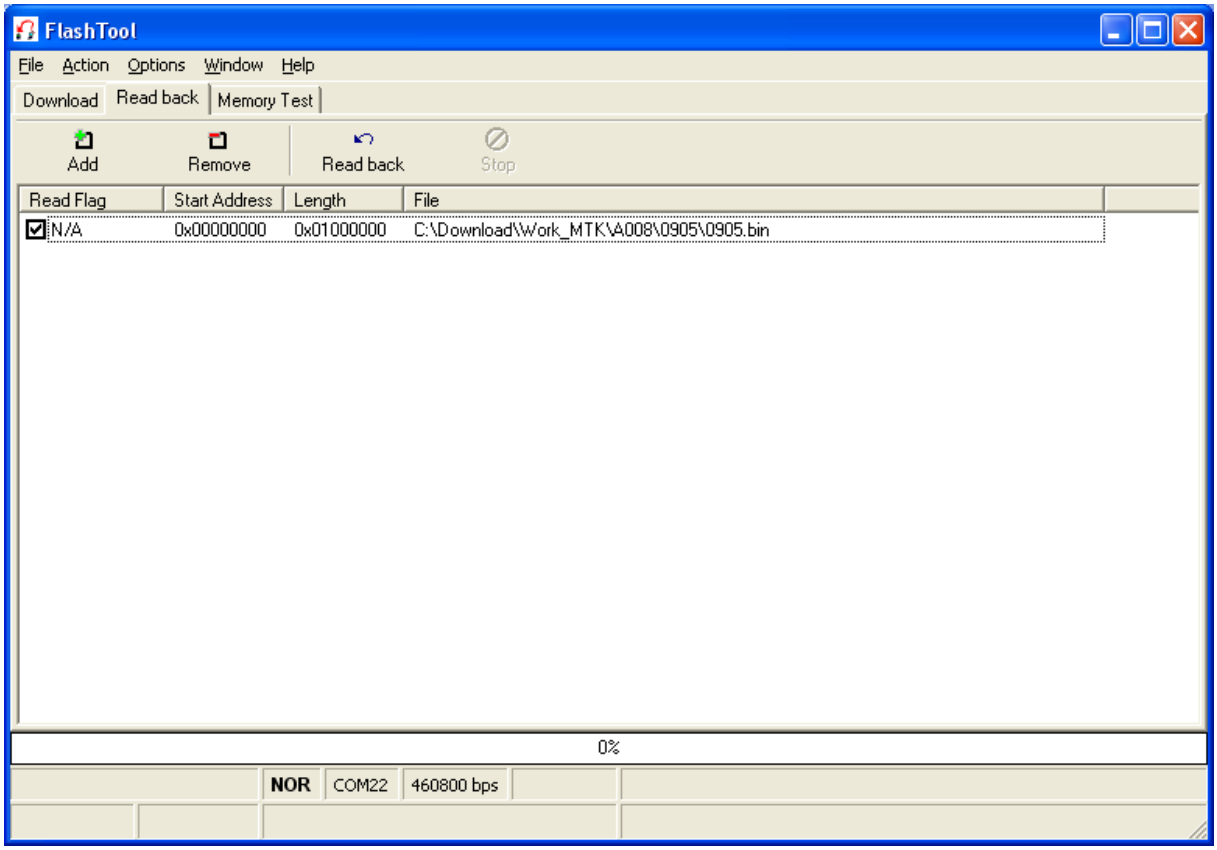
Flashing in practice, using MTK flashtool

MTK FlashTool is the "official" tool used to flash MTK phones. It's used by the factories and developers. It is also where the DA originates from.
The features of MTK FlashTool include

- Reading(Read-back) and Writing(Download) the flash
- Formatting the File systems

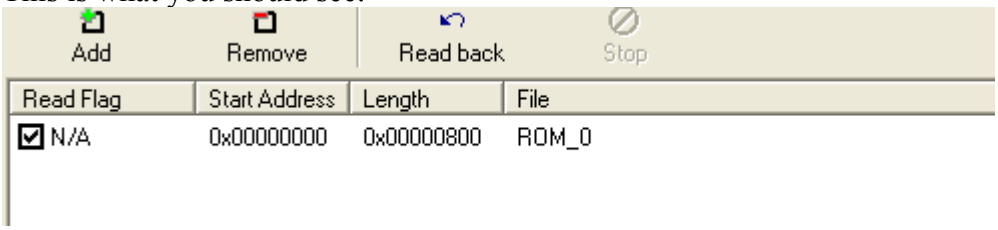
There are video guides available even on using this trusty piece of software, but I will attempt to go over the basics in writing.

Reading a full backup with MTK FlashTool



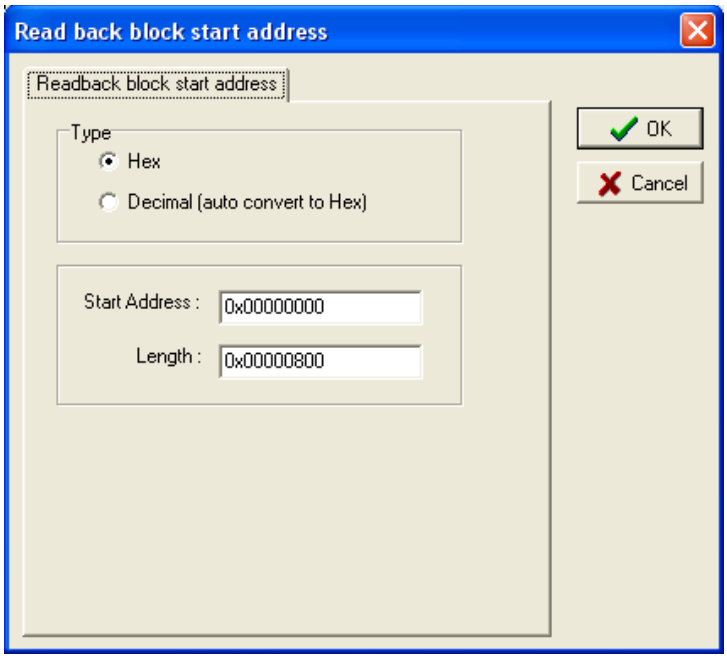
This is MTK FlashTool. You can set your Serial Port settings in the Options menu.
If you're using the Direct COM interface, select COM1/2 and 115200 bps from the Baudrate submenu. Otherwise select the virtual com port number of your USB serial cable, and 460800. Also set Options->Operation Method->NOR (this book only talks about NOR flash)

Click over to the Read Back tab, remove everything you may find in the list, then click Add. This is what you should see:



Now double click on ROM_0, and Select where you wish to save your image.

Then you will be greeted with the dialog below



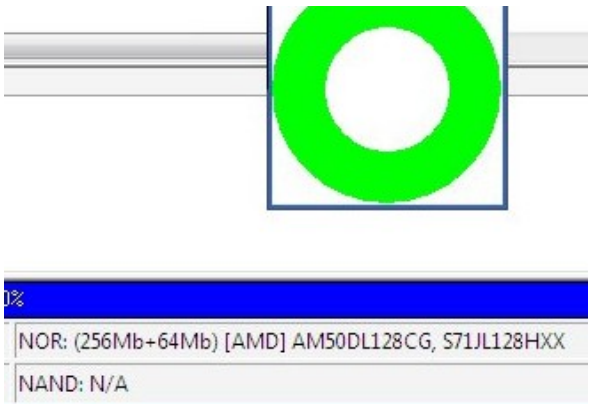
It's fairly obvious what we need to do here. For a full backup we will start at address 0x0000000, this means the very beginning of the flash chip. The length has to be equal the size of our flash chip. Don't know this? Try reading a small chunk first, and I'll show you how to find it.

Click Ok, and Click Read back. At this point you should have your phone physically connected to the PC, with it being switched off. Push the power button. Your phone should start booting the DA, nothing will appear on most models while doing this. Some will turn on the back light, some will produce a hissing sound through their speaker. These are all normal.

If the phone turns on and starts the firmware, then your interface, settings are wrong/faulty ☺ If you've used the Multimeter method now would be the time to try the next possible combination. If it just doesn't seem to work in any combination, you should loopback test your interface cable. Or it's also possible that your phone doesn't have UART on it's data cable connector, only on test points, at this time I'm not going to write about finding the right test points on such phones.

Try and try again, until it works. If you've read your first backup, Congrats!

Ok, now if you didn't know your flash chip size the bottom right corner is where you should look, after your initial smaller readback.



You're interested in NOR: (256Mb+64Mb)
What this means is that this phone has NOR Flash, 256 MegaBITs of it, and 64 MegaBITs of RAM.
Now you can use Google.
Use the following search without the quotes, and substitute the number you got:
"256 *131072 in hex" The result in this case:

256 * 131 072 = 0x2000000

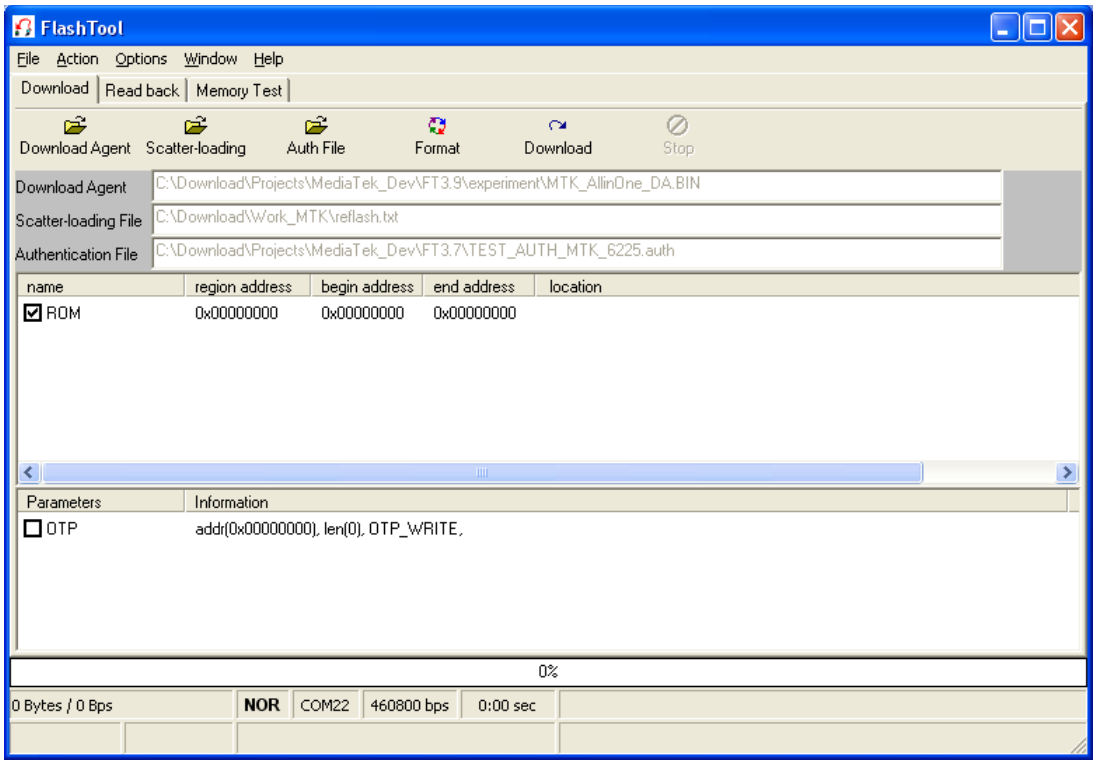
0x2000000 is what you would need to enter in the length field to get a full backup of this chip.

As a quick reference:
128 MB = 0x1000000 (usual file system start: 0xE00000, length:0x200000)
256 MB = 0x2000000 (usual file system start: 0x1C00000, length: 0x400000)
512 MB = 0x4000000

Full Flashing with MTK FlashTool

Full flashing, with no regard for the Security Zone, as discussed above, this will create bricked phone unless you continue with the the fullflash guide, have an image with an open Security Zone, or one with Security Zone protection disabled.

First open notepad (that's right notepad) Insert the following line:
ROM 0x0
And save it as reflash.txt
Now in FlashTool click the Download Tab.
Click the Scatter-loading button, and locate your reflash.txt
This is what you should see:



Double click ROM, and select the image you wish to flash to your phone.
Hit Download, and press the power button on the phone. Sit back and enjoy.

Patch flashing with MTK FlashTool

This is really similar to full flashing, with only a few differences. You will need to create a scatterloading text file.

The format of this file is simple:
NAME ADDRESS <SIZE>
NAME ADDRESS <SIZE>
NAME ADDRESS <SIZE>
Etc..
<SIZE> is entirely optional

FlashTool will automatically load NAME, if it finds it in the same directory as the scatterloading txt. For example you wouldn't need to open the SECAR file from the full flashing guide manually; it would automatically add it when the scatterload txt is opened. One more thing to do when patch flashing is disabling "Options->Baseband Chip Option->Check Baseband chip HW version". This is because FlashTool will try to guess what chip the firmware is for, from the flash image, however it will fail to do so if you're just patch-flashing random data, and will refuse to start the flashing procedure.

A valid scatterloading file may look like this:
LOC1_6642A4 0x6642A4 0x03E8
KEY_668324 0x668324 0x0274
DICTIONARY 0xA41A9C 0x1FB7B4
MLTABLE 0xC3DA14 0x0150
LTTABLE 0xC6257C 0x0498

LOC2_CF60F8 0xCF60F8 0x03E8
MAPPING 0xCF787C 0x549A0
CAL_D6DE70 0xD6DE70 0x0010
BLD_DADF04 0xDADF04 0x0010

There is a rule that patch-flashing starting addresses have to be DWORD aligned. What does this mean? If you wish to patch flash at 6642A5, it won't allow this, as 6642A5 isn't on a DWORD boundary. DWORD stands for double-word it means a 32 bit integer, which means four bytes. So basically, the starting address has to be a multiple of 4. The multiple of 4 closest to 6642A5 would be 6642A4, so you would have to add that 1 byte of "padding" even though the data you wish to patch starts 1 byte after. (Security zones conveniently start at multiples of 4 ☺)
How to check if an address is DWORD aligned by using Google:
Search: "0x<insert your hexadecimal address> mod 4" without quotes.
Example:

0x2DB6C0 modulo 4 = 0x0

If the result is 0x0, then the address is DWORD aligned.

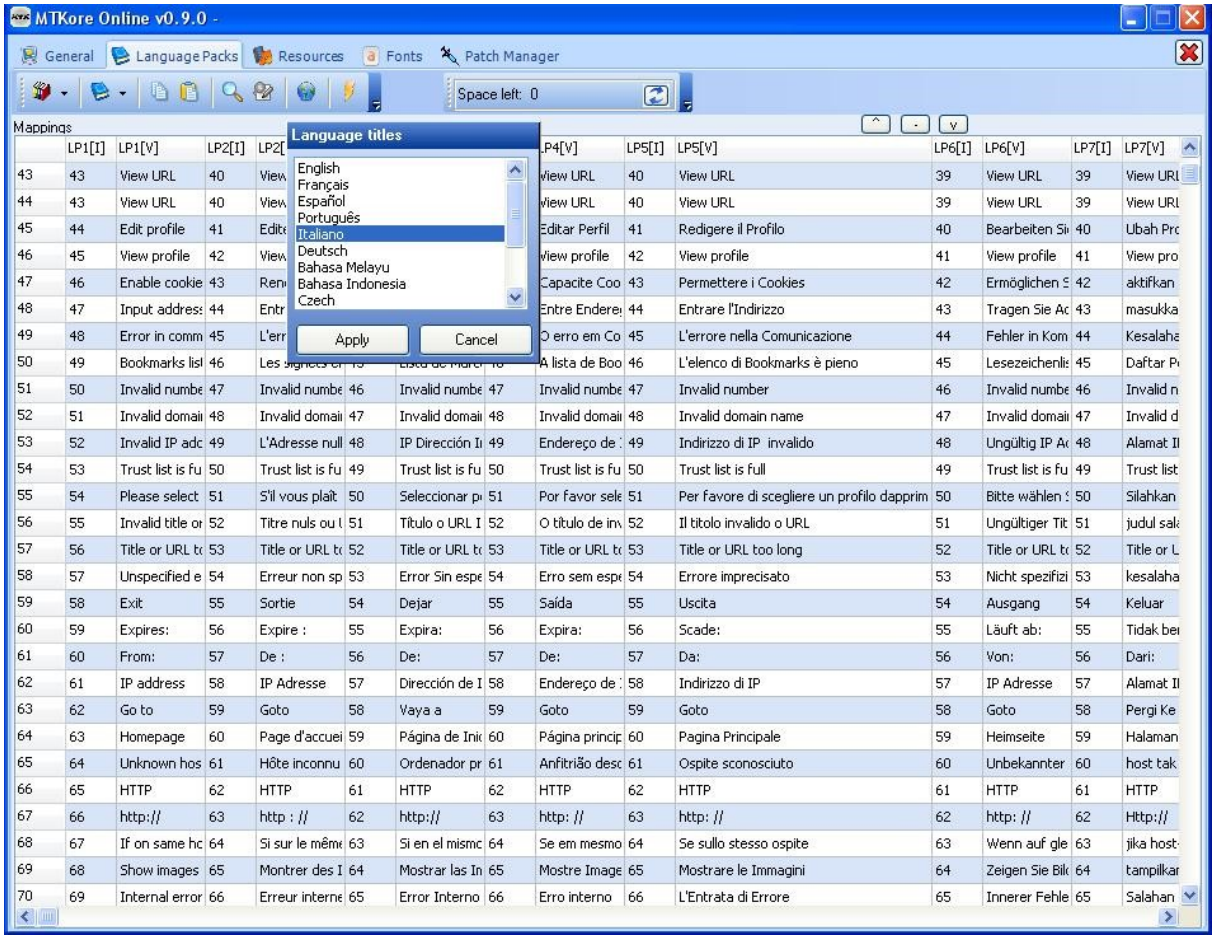
A word regarding USB Download option: No you still can't use the phone's own USB Data Cable ☺ this is an option that simplifies detecting the interface used in factories. It is a simple FTDI based USB Serial Cable, with a unique ID that it is looking for, if you enable this. Sorry…

Firmware Editors

Now that you've gotten a pretty good idea about how to interface with your MTK phone, and the caveats of flashing we come to the subject of firmware editors.
Firmware editors allow you to change internal resources of MTK firmwares, like language packs, graphics, internal ringtones, and fonts.

Most boxes that support MTK phones will have SOME kind of firmware editor. The quality of these varies from box to box: there are some good ones, but there are also 15 minute hack-jobs that fall in the "because firmware editor looks good on the marketing pages" category, we're not going to discuss either of them here. The two boxless firmware editors we'll take a look at is our very own MTKore (After all that useful information, some shameless self promotion is in order ☺) and z3x team's ChinaEditor.

PhiKore – MTKore Online



MTKore is our firmware editor product. It's in the early stages of evolution, currently looking for Beta testers. It supports **language** pack editing, **resource** and **font** modification. For our software protection we decided on a client/server structure, this allows us more freedom in processing code development, without having to issue updates for the client software, and allows us to sell our software purely virtually. You do not need to wait for boxes or hardware protection dongles to arrive, to start using MTKore, you only need to sign up for our beta, and wait for a download link.

MTKore has smart dictionary management and can build your dictionary as you translate. When applying a dictionary you'll be asked if you wish the items that haven't been automatically translated to be highlighted for you. Our focus is on making the use of the software as pleasant as possible for the user.

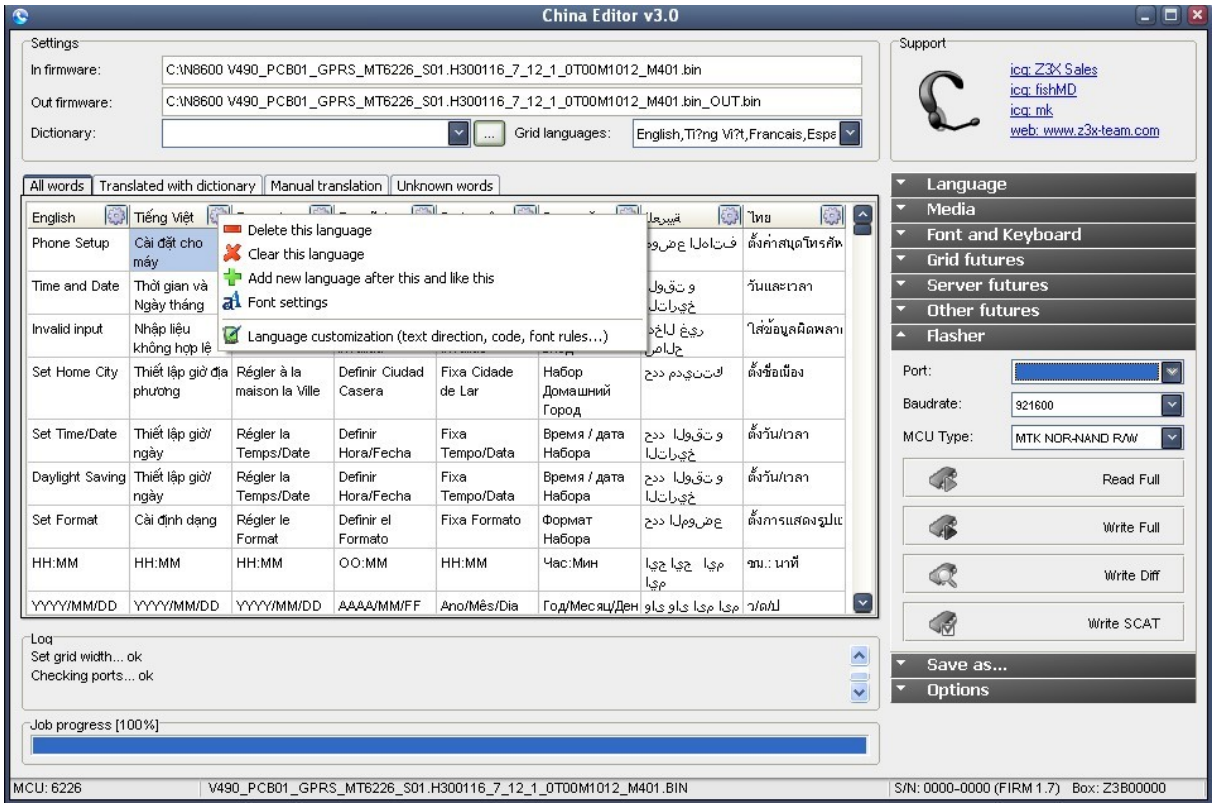
Beta signup fee is 50 USD (this also includes a live account, when MTKore leaves Beta)

Contact: betatesting@phikore.com or phikorepublic@yahoo.com or phikore@gmail.com
Guaranteed answer in 24 hours.

URL: http://www.phikore.com

team z3x – China Editor

China Editor is an extremely versatile piece of software that supports MTK and Spreadtrum firmwares. It has LP, Resource, Font and now Keyboard editors. It has been in development since roughly about the same time as MTKore, however it has gone public way sooner, so is a more mature product. The protection scheme is hardware key based. They will ship you a USB dongle that allows you to use the product.



China Editor's price including hardware key is 130 USD + Shipping

I've seen great translations made with this software, so I'll have to assume that It's good usability wise. I can't offer a more in-depth review as I haven't personally used it.

Check them out:

URL: http://z3x-team.com/

A quick comparison:

Feature	MTKore	China Editor
Language editing	X	X
Resource editing	X	X
Font editing	X	X
Keyboard editing	planned	X
Integrated flasher	X	X
Export to FlashTool scatter	X	X
Dictionary handling	X	X
Advanced language customization		X
NAND support	planned	X
Spreadtrum support	Separate product in the future	X
Import export of dictionaries and language packs to CSV	X	X
BitText support	X	
Hardware Key dependence		X
Internet connection dependence	X	

Above review of China Editor is based solely on my views and opinions.

This is is for the time being. If you have any questions, or comments please direct them to: phikorepublic@yahoo.com