

# *PPCGeeks Kitchen Tutorial*

*Revision 1.0*



This Documentation is going to assist you in a number of things regarding the Apache Device Kitchen. Specifically we are going to be focusing on the “New” Core. This kitchen functions differently than those previously released, as this is going to be the new foundation for what we will see in future kitchen releases, I am going to explain in great detail how it works, and how to get it to work for you.

On the next page you will find the table of contents. I am going to walk you through the very basics, and into the extremely complex. You may of course skip ahead to the section in which you are seeking assistance. If you do NOT own the device pictured below, you very well may have downloaded the wrong tools for what you are seeking to do, and the probability of you “bricking” or making dead your handset has just increased ten-fold.



This... is the HTC Apache. It is quite a spectacular piece of engineering. You'd never know it until you upgraded it, but there it is in all it's splendor. This device shipped with an early revision of the Windows Mobile 5 architecture, known to enthusiasts as AKU 2. The first “hacked” update to this phone brought the AKU up to 2.2. Further reverse engineering has allowed us to see upgrades as far as 3.3, and even an *intensely* hacked 3.5, those 3.3 and 3.5 revisions however, were built upon the core foundations of the 2.2 “CORE”.

Thanks to the efforts of many, a “True” AKU 3.5 core has been brought to the surface, and that is what we will be working with today.

*SECTION I:*

PG 4

What is a kitchen and what can it do for me

- What is a Kitchen
- Where do I get the Kitchen
- What do I do with these files
- Folder by Folder breakdown

*SECTION II:*

PG 8

How do I make packages to add to the kitchen

- How to convert your CAB files into OEM packages
- Generating Registry Entries for your OEM application
- Creating shortcuts and managing CLSID
- Generating an option.xml file amongst other things
- Complicated OEM packages
- Creating sub-directories and moving files into them

*SECTION III:*

PG 28

Making and burning your first ROM

- Running BuildOS and CreateOS
- What is in the “temp” folder
- What to do with the nk.nba file
- Patching in your own custom boot screen
- Converting the nk.nba file into a flashable nk.nbf
- Flashing your new ROM onto your handset
- BigJ’s tips & tricks
- 

*SECTION IV:*

PG 38

When it all goes wrong

- Common Error messages and what they mean
- Stuck in Bootloader Mode
- White Screen of Death
- Hanging on the Bootscreen
- Hanging on the second splash screen
- Locks up at the today screen

## Section I: What is a Kitchen

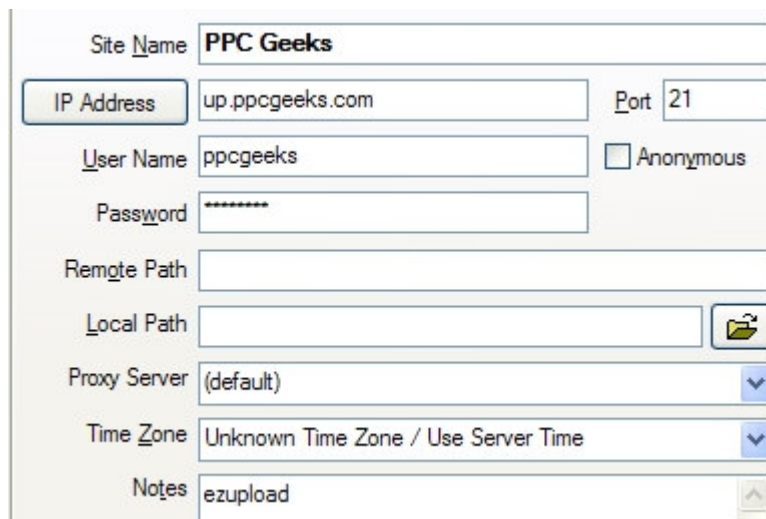
### *What is a Kitchen?*

A Kitchen is a term used for the set of tools used to create or “cook” a ROM for your phone. They typically include a large set of confusing folders and files that would make no sense to anyone outside of the “know”. These files and tools are used to create an operating system for your handset, specially customized to suit your needs, as well as allow you to upgrade your operating system files outside of your Carriers designated upgrade path.

### *Where do I obtain a ROM Kitchen?*

PPCGeeks has an FTP where it stores the numerous kitchens available for the HTC Apache.

FTP: up.ppcgeeks.com  
PORT: 21  
USER: ppcgeeks  
PASS: ezupload  
FOLDER: /Apache/



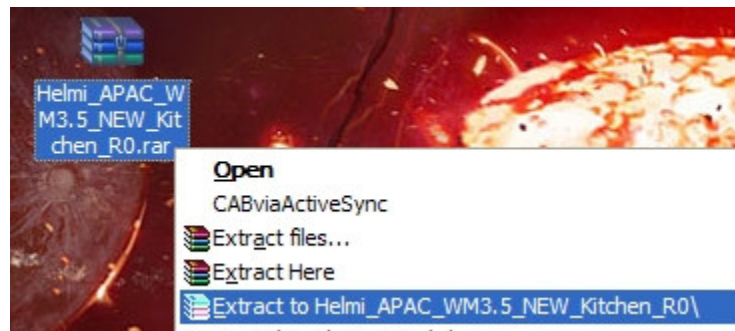
The image shows a screenshot of an FTP client configuration window. The fields are as follows:

- Site Name: PPC Geeks
- IP Address: up.ppcgeeks.com
- Port: 21
- User Name: ppcgeeks
- Anonymous: ☐
- Password: (masked with asterisks)
- Remote Path: (empty)
- Local Path: (empty)
- Proxy Server: (default)
- Time Zone: Unknown Time Zone / Use Server Time
- Notes: ezupload

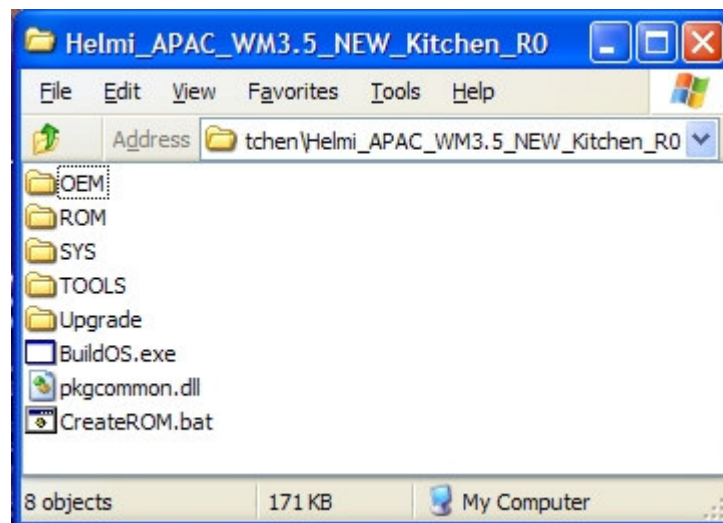
Once in the /Apache/ directory you should have no trouble locating the most recent version of the PPCGeeks kitchen. There you will also be able to locate various prebuilt kitchen OEM packages created by other members of the community. I would recommend investigating those prior to using however as you will more than likely find errors or inconsistencies.

***What do I do with these files now?***

Now that you have downloaded the kitchen as a single file archive (zip or rar). You will need to extract it to your hard drive. Most systems have an archive extractor built in. For those of you that do not, I will need you to go to [www.rarsoft.com](http://www.rarsoft.com) and download the latest trial version of “WinRAR”. This application will allow you to get to the files contained in the archive. Once you have installed WinRAR, simply right click on the downloaded archived kitchen file, and choose the option “Extract to FOLDER”.



You will now have a folder that holds the files and tools of the ROM Kitchen. It should look something like this.



In this state, you could simply click on BuildOS.exe, build the OS, CreateROM.bat and make the nba file, convert that to a flashable nbf, and flash. Bam your done... simple right? Well yes and no. Yes in that you will have all of the benefits of an updated operating system. No in that you have not yet learned anything and therefore not customized your handset to suit *your* needs. In order to do this lets explore the folder structure you see here.

### ***Folder by Folder breakdown***

#### **ROM**

This folder holds the skeleton nba file that later is used as a reference for building your own ROM during the CreateROM process. This folder should not be touched by you.

#### **SYS**

This folder holds all of the necessary files to make and build the operating system. This is where you will find all of the base functionality, 90% of these files and folders are mandatory and you will not be able to make many, if any changes at all to this directory. If you *REALLY* know what you are doing then have at it, if you are not comfortable with manipulating root operating system files, then you should not touch this directory either.

#### **TOOLS**

This folder has all of the tools that are used during the ROM creation process. The \*.bat file “CreateROM.bat” calls upon a large majority of the files in there to manipulate the kitchen data once it’s been put into a dump. If you mess with anything in here, your ROM may not build properly. This folder should also not be touched by you.

#### **UPGRADE**

This folder has the files you use to actually flash your handset once the ROM has been created successfully. I think this folder has its uses, but later we will learn what we can and can’t change in there. For now, well just leave it alone. In a nutshell, I would move over the new flashing exe “APACUpgradeUt\_noID.exe” and use that one. This will absolve you of having “COUNTRY ID ERROR” when trying to flash your newly built ROM.

#### **OEM**

Now we come to the folder of massive manipulation. This folder holds your customizations. Any type of addition that you wish to build into your ROM, go into here. You will see a few folders in here that you will not be allowed to mess with, such as the Apache Drivers. Moving files into SYS is not necessary so to speak, but it allows you to organize your kitchen better, as well as not have to worry about accidentally manipulating a folder or file that will cause devastation to your system. Here is a list of what is not listed as optional in this folder.

#### **APACHE**

-drivers -sap -service -setting

#### **CERT**

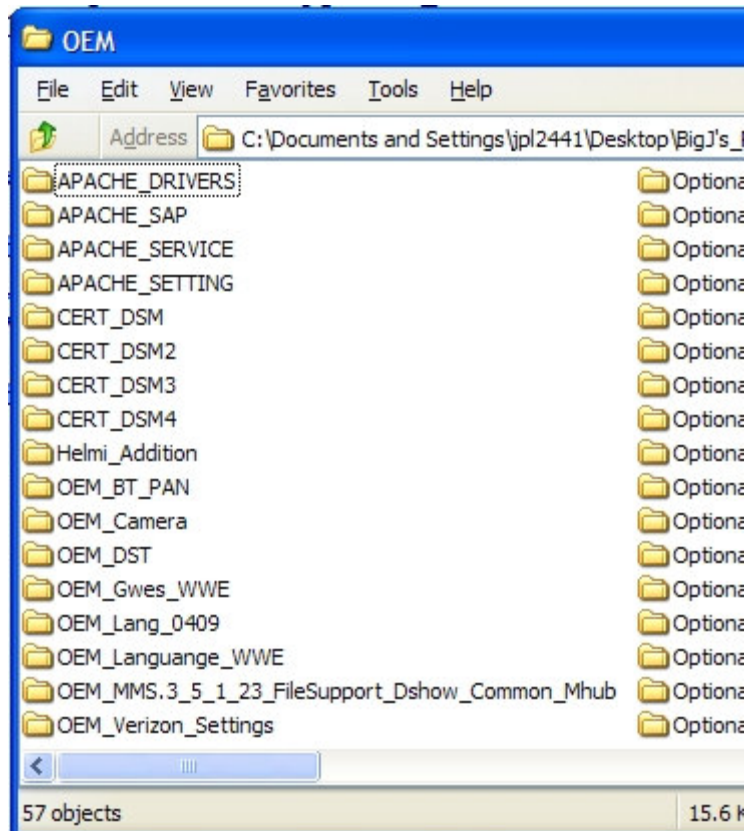
-dsm -dsm2 - dsm3 -dsm4

#### **OEM**

-btpan -camera -dst -gwes -lang -langwwe -mms -verizon

#### **HELMI**





Looking at this list we can easily see what we simply *cant* build our ROM without, and what should be listed as optional. So lets rename what should be optional as such, and move the rest over to the SYS folder. Add the word “Optional\_” to the front of Verizon. Verizon is not mandatory, and should be kept in here as optional.

Now let’s move the rest of the non optional folders to the SYS... Highlight all APACHE, CERT, HELMI, and remaining OEM and move those to the SYS folder. Now you should have a folder loaded with “Optional\_”.

The Optional\_xxx are folders containing what is commonly reffered to as “Kitchen OEM”. These are programs, settings, pictures, ringtones, etc. that have all been converted into a format that can be built into the phones operating system *as a native application* in your handset. This means that even after a HARD-RESET, these applications and settings are retained. The next step is to learn how to turn all of your favorite programs into “Kitchen OEM” packages.

## Section II: Creating your own OEM's

### *Converting & Working with CAB files*

First we need to do a little footwork. We need to locate all of the CAB file installer versions of all of the applications that you wish to turn into OEM packages. This is simple for some as most applications offer a CAB as a download option. As you see here, the top download is an exe installer, but just beneath you can choose to download the corresponding CAB file. Some will offer an installer only, but they merely dump the cab file into "C:\Program Files\Microsoft ActiveSync".



[v1.60.2 Lite trial version  
and upgrade](#)  
[v1.60.2 Full CAB file](#)  
[v1.60.2 Lite CAB file](#)

Others are much more devious and package the cab files into exe installers which either hide the CAB file somewhere on the computer in places such as "C:\Documents and Settings\Administrator\Local Settings\Temp", or not only hide it, but will DELETE it if the phone is not connected to the computer, forcing you to run the exe file again when the handset is connected. Just remember, it has to be on your computer somewhere, and if it's on your computer, you can find it. I have done creation date searches on my machine for some of the more elusive, while having my phone hooked up, just not hitting OK on the device to suspend the deletion of the CAB file. Regardless of how you obtain them, you need them to build the OEM package.

### *I've obtained my CAB files, now what?*

Now that you have obtained the cab files, you need an application that lets you break them down. I use WinCE Cab Manager personally so that is what were going to use today. A slightly older version 21035 has a crack (look real hard), the newer version 21050 however has yet to be cracked. It requires windows scripting host 5.6 to be installed to work properly, just punch that into google and grab it. Now that were all setup your CAB files should look like this...



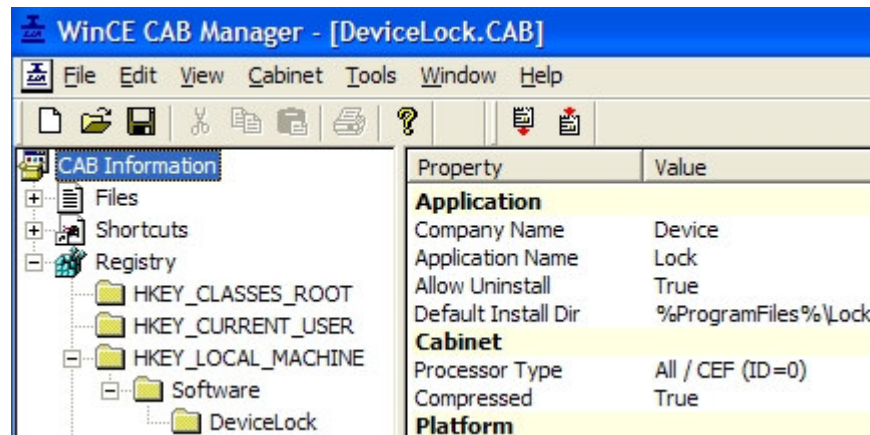
Double click on that file and it will load into a window. For this example we will do an extremely simple cab file, in our next example we will do a more complicated one. Our first example will be a program called DeviceLock. This application does not need a shortcut, *BUT* the registry for this application has an issue we will need to resolve. Well go through this step-by-step to ensure you are comfortable with breaking open your cab files.



1 – Double Click your selected cab, in this case “DeviceLock.CAB”



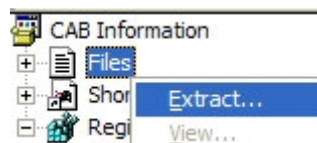
2 – It will open up exactly as shown below



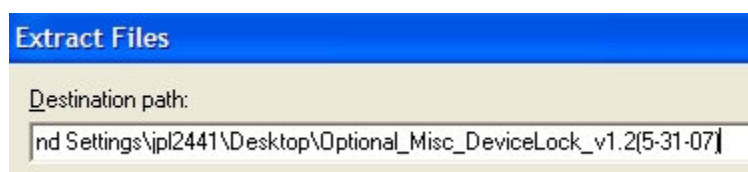
3 – Click on the “FILES” and look at the list of files. Pay special attention to where those files are directed to go. Any file that says %InstallDir% are slated to go into the Default Install Dir listed in the screenshot above. In this case “%ProgramFiles%\Lock”.

CAB Information		Index	Name	Location	Type
+	Files	1	DeviceLock.exe	%InstallDir%	Application
+	Shortcuts	2	DeviceLockCPA.cpl	%Windows%	Control Panel extension
+	Registry				

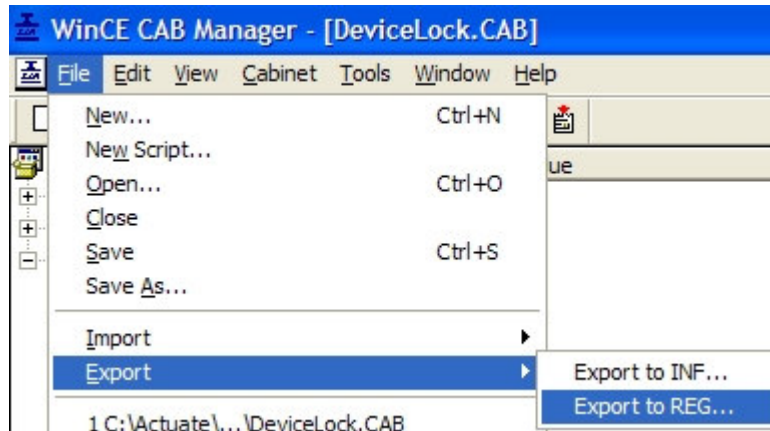
4 – Since we are going to force the %InstallDir% to /Windows/ and the other file is going into %Windows% already, this is what makes it a simple CAB. Now Right Click on the files and choose to extract the files.



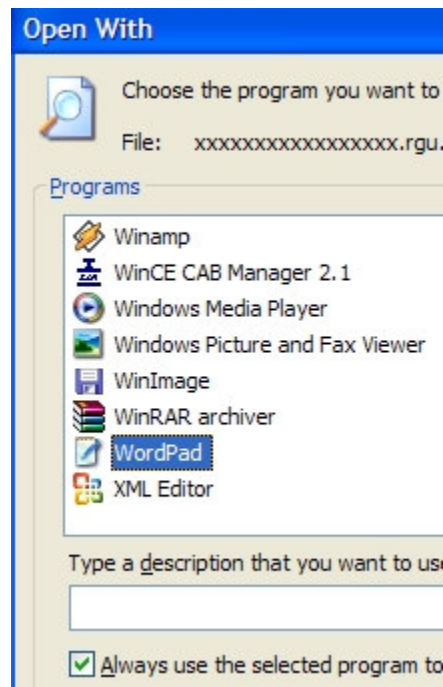
5 – Name the directory in which to extract the files. Lets just make it simple and name it according to the new kitchen OEM scheme. Optional\_ProgramName\_Revision(Date). In this case it will be Optional\_DeviceLock\_v1.2(5-31-07). This way not only can you track the revision of the application package you are making, you can track your changes according to date



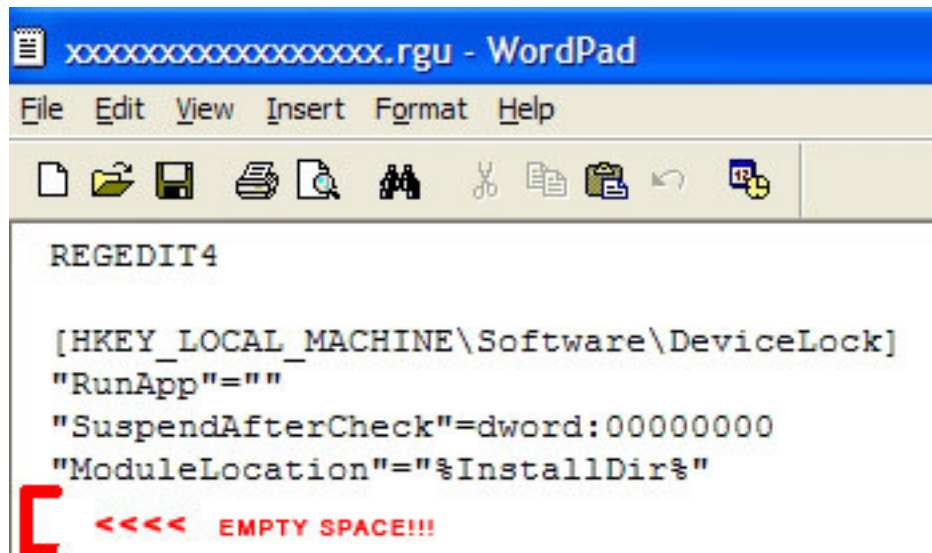
6 – Now its time to extract the registry information for this program. This step is slightly different. Highlight the registry in the list. Now go to FILE, and EXPORT, and EXPORT TO REG. Name this anything you want, as the name will change later, for simplicity I typically name it xxxxxx to remind me to change it later.



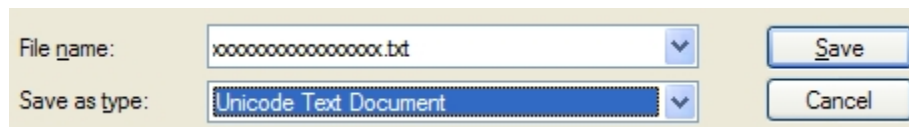
7 – One VERY important and often overlooked step comes next. You will need to convert the exported registry into UNICODE. You will also need to make sure there is an EMPTY line after the very last entry. Now there are several ways to do this, my own is radically different than what I'm going to teach you, but it is effective none-the-less. Rename the file extension from .REG to .RGU. Now double click the file, it will ask you what program to open the .rgu file with. Choose "WordPad". (not notepad), and go ahead and permanently allow that file extension to open with WordPad.



Once you have the .rgu open in wordpad, make sure you have the empty space at the bottom.

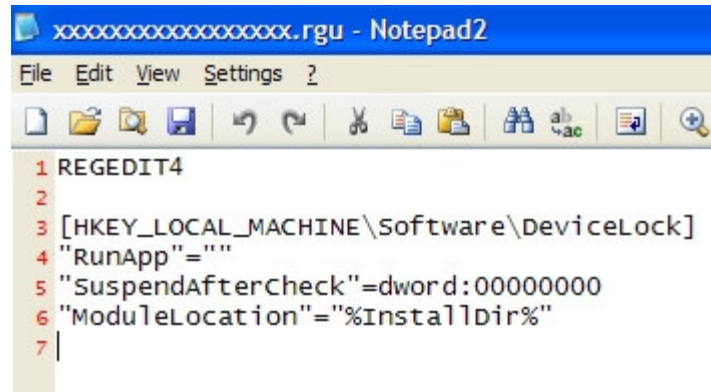


Now save the document as a “UNICODE” document. Maintain the current file extension of .rgu as it will default to .txt as in the picture below. It may ask if you want to overwrite the current file, choose Yes.



8 – Now we have the registry file in the correct form to work with it. This is the time you will go through EVERY SINGLE entry to ensure that they are formatted properly so that you don’t get errors when you run BuildOS.exe. The older kitchen would tell you wich rgu file was the problem and which line... the new one however does not, and it makes troubleshooting a bitch, so do it right the first time. That being said, I have a screenshot of the program I use to edit files ( I LOVE line numbers, the program is called Notepad2 and is available here.... <http://www.flos-freeware.ch/notepad2.html> and is free). If you do choose to use this program to associate rgu files with, to encode in unicode you simply go to the FILE dropdown and choose ENCODING, and just click UNICODE and your done. From here on out, all of my screenshots will be done using that program.

Anyways, here is the registry entries we are working with.

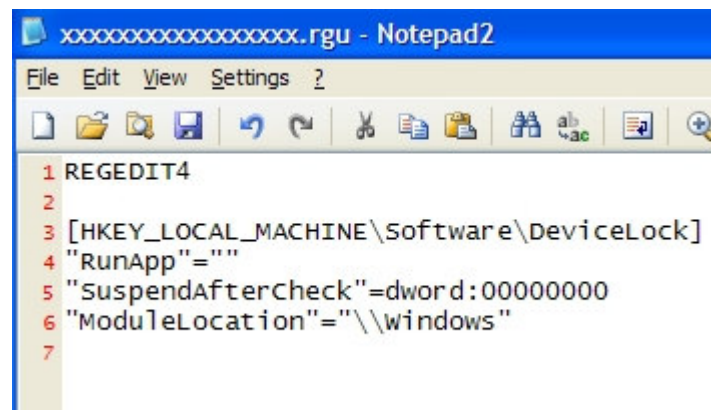


```

1 REGEDIT4
2
3 [HKEY_LOCAL_MACHINE\Software\DeviceLock]
4 "RunApp"=""
5 "SuspendAfterCheck"=dword:00000000
6 "ModuleLocation"="%InstallDir%"
7

```

The problem is on line 6. You see a cab file translates things with %'s into what it is supposed to be, an OEM package will not. So we need to change this to point to the windows directory. You would think that the entry would change to \Windows .... But it does not. The entry needs to change to [\\Windows](#). When the OS builds the registry it needs to have double "\\" in order to process properly. When you have extremely large registries you have exported, you have to look very carefully for entries made with a single "\" and entries made with "%" (relative paths). Once corrected it looks like this, and you can now close the file and not have to worry about it until later (later we rename it).



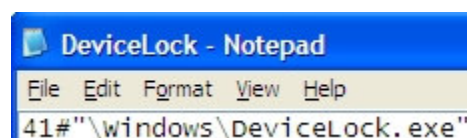
```

1 REGEDIT4
2
3 [HKEY_LOCAL_MACHINE\Software\DeviceLock]
4 "RunApp"=""
5 "SuspendAfterCheck"=dword:00000000
6 "ModuleLocation"="%\\windows"
7

```

9 – Even though this OEM does not *\*need\** a shortcut, lets make one anyways for the purposes of this tutorial. Make a new text document and name it what you would like the shortcut called... DON'T use spaces, you can add spaces to the name when the OS builds. This shortcut will be called "DeviceLock.txt". Open it up. Now were going to pick an arbitrary two digit number, 41, that's a good number because its somewhere in the middle. Now the shortcut format needs to be EXACTLY as it is below

41#"\\Windows\\DeviceLock.exe"



```

DeviceLock - Notepad
File Edit Format View Help
41#"\\windows\\DeviceLock.exe"

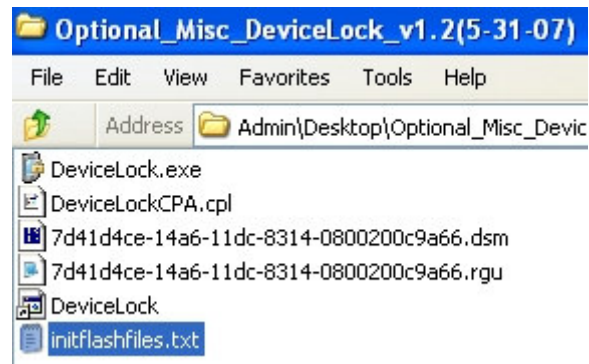
```

Now save the file, leaving the default encoding of notepad which is ANSI. Converting the shortcut to Unicode will break it. Once its saved, rename it with the extension “.LNK” once you do this the file will turn into a windows logo and have the little arrow denoting a shortcut.



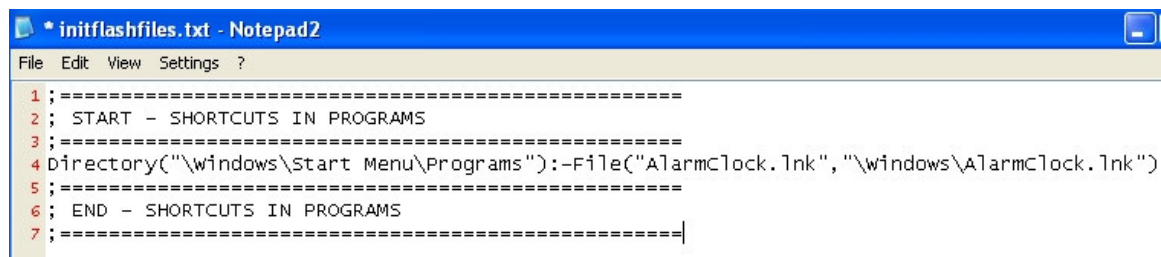
I couldn't tell you the purpose of the number, and the directory should be obvious. The only thing you will need to manipulate for every single shortcut you make is the name of the exe, in this case its DeviceLock.exe (exactly as the exe is, including caps). Its case sensitive, and you can only shortcut ONE exe at a time. So if you need more than one shortcut, you need to make more than one shortcut file.

10 – Now comes the tricky part. You need to make a file (yes its getting redundant) that will tell BuildOS.exe where to put your shortcut, like do you want it in your start menu under programs... or is it a game and you want the shortcut to your game in the games folder where solitaire resides? We're going to make another txt document and name this one “initflashfiles.txt”



Open the initflashfiles.txt, copy & paste the text below exactly as it appears to make your shortcut appear in the start menu, under programs,

`Directory("%Windows\Start Menu\Programs");-File("DeviceLock.lnk","%Windows\DeviceLock.lnk")`





Notice in the picture that there is **NO SPACES** (aside from the “Start Menu”) so when you copy and paste, make sure it doesn’t arbitrarily add a space after the :-

Another thing you may notice is that I have commented both the START and the END of the initflashfiles.txt this is considered “Best Practices” and is not necessary, I assure you however if you do not do this, a complicated cab file will give you nightmares trying to troubleshoot if you don’t comment the start and end of what you are having the file write. As a side note, if you do comment a file, you *MUST* end comment the file as well or it will not get processed by BuildOS.exe. If you want to make it simple you can just end with a semi-colon; as in the below example. Please don’t be lazy, follow the above ex.

```
* initflashfiles.txt - Notepad2
File Edit View Settings ?
1 ; START - SHORTCUTS IN PROGRAMS
2 Directory("\\windows\\Start Menu\\Programs"):-File("AlarmClock.lnk", "\\windows\\AlarmClock.lnk")
3 ;
4 ;
```

11 – Great Everything looks ready to go. Now all we need to do is assign this OEM package a unique identifier so that the OS can build a CLSID for it. Without a unique ID this package will never build into your ROM. You can generate unique ID’s at this URL <http://www.famkruihof.net/uuid/uuidgen>

I would generate about 300 or so and just save them to a txt document on your computer somewhere for future use. As you use them, be sure to delete them or mark them out so you don’t accidentally use the same one again. Lets generate some right now.

The following version 1 UUIDs / GUIDs are generated for your use:

```
7a16a24a-14a6-11dc-8314-0800200c9a66
7a16a24b-14a6-11dc-8314-0800200c9a66
7a16a24c-14a6-11dc-8314-0800200c9a66
7a16a24d-14a6-11dc-8314-0800200c9a66
7a16a24e-14a6-11dc-8314-0800200c9a66
7a16a24f-14a6-11dc-8314-0800200c9a66
```

Highlight the top number and copy it. Now go into your OEM folder and right click and choose to make a new text document, name this new document

7d41d4ce-14a6-11dc-8314-0800200c9a66.dsm

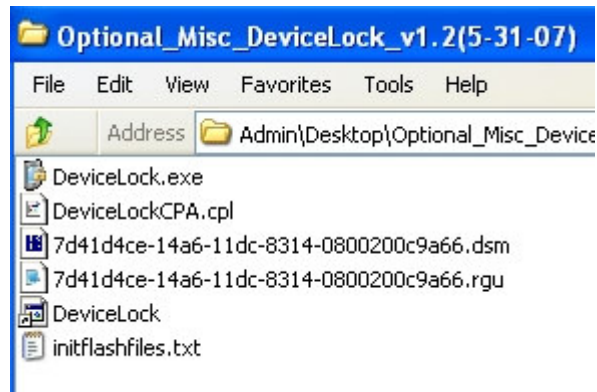
Remember the DSM extension and not .txt like you would think.

Now RENAME your xxxxxxxxxx.rgu to the exact same thing, save the file extension so it will look like this ...

7d41d4ce-14a6-11dc-8314-0800200c9a66.rgu

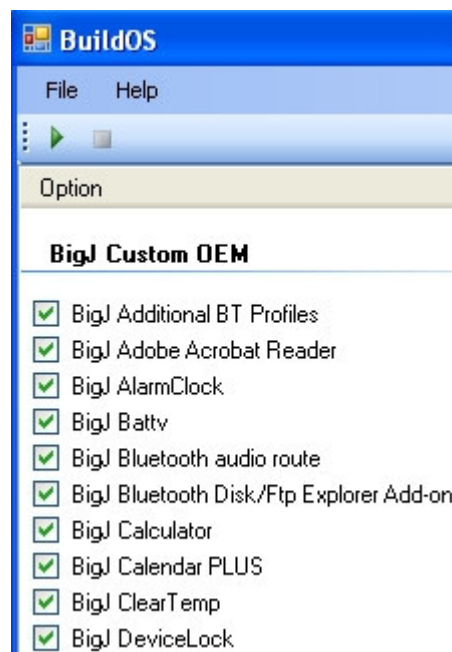
your OEM package folder should now look like this





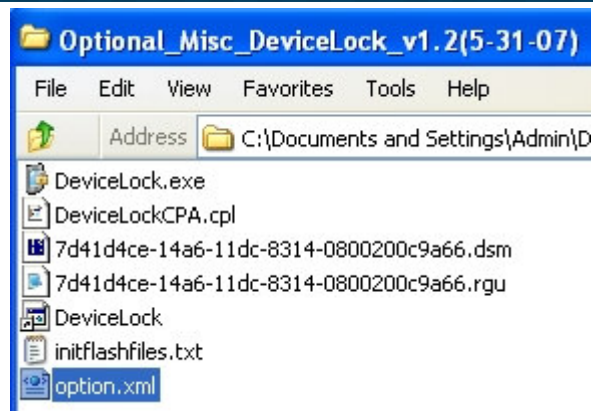
Previous kitchen OEM packages would now be 100% ready to be dropped into the OEM folder and built. The new kitchen however requires one additional step to take place before its ready...

12 – Now we need to get your package to show up on the list with a cute little checkbox when you open up BuildOS.exe.

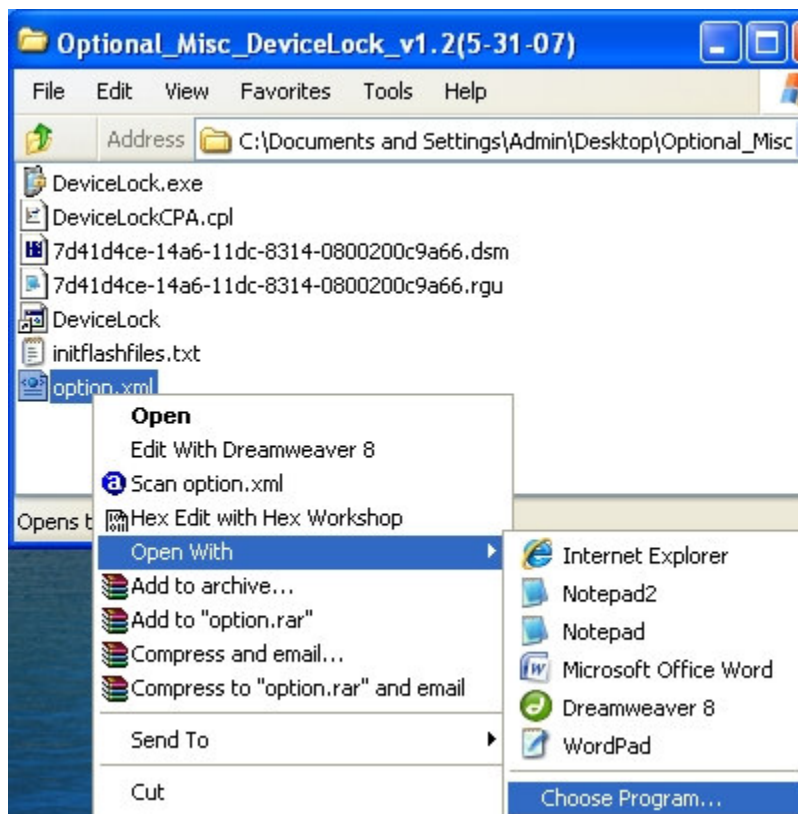


As you can see from this screenshot, I have designated a category for this OEM package to show up under “BigJ Custom OEM”. You can also see that it is checked off by default for building into the ROM. This is what we are going to do next.

Guess what? Make another txt document and name this one “option.xml”



I don't know what you have associated with xml files, but change it to Notepad2 (the program mentioned earlier that we associated everything else with). To do this, right-click the file and choose the option OPEN WITH, and then CHOOSE PROGRAM...



As we have previously, browse to Notepad2.exe and choose to permanently open files of this type with that program. Once open you can then follow this relatively simple as there are line numbers here for you. Im going to number the lines on the left, but they are not part of what you need to paste in there. The screenshot will help you to understand

```

1 <?xml version="1.0" encoding="UTF-16" standalone="yes"?>
2 <Items>
3   <Item name="BigJ DeviceLock" group="BigJ Custom OEM" checked="true">
4     <Tip>Enhanced LOCK functionality with settings</Tip>
5     <Guid type="p">7d41d4ce-14a6-11dc-8314-0800200c9a66</Guid>
6   </Item>
7 </Items>

```

**Line 1:** tells the document that it is an XML document, never change this

**Line 2:** tells the xml document that you are adding an item to the list

**Line 3:** this line is important so were going to break it down section by section

- name= This is the name of the OEM package as it will appear on the list of programs when you are looking at BuildOS.exe, in this case “BigJ DeviceLock”
- group= This is the CATEGORY that your OEM will appear under in the BuildOS. In this example it is “BigJ Custom OEM”
- checked= This is where you tell BuildOS if your OEM package will be checked off by default

**Line 4:** this line is also important especially if you are going to be uploading your OEM packages to share with the community. This line allows you to type in a custom tooltip that will popup when users hover over your package in the BuildOS utility. It’s a good idea to explain what your application is, possibly its revision, and anything else that may be useful to a stranger. This tip can be as long as you would like, so long as you keep it ALL on line 4. In this example the tooltip is “Enhanced LOCK functionality with settings”

**Line 5:** this is where you will copy and paste your CLSID that we generated for this package. This tells BuildOS that this folder should or should not be built into the ROM. So maneuver back over to the DSM file, copy the CLSID and paste it in there. You don’t need the files extension “.dsm” just the numbers & letters prior to that. Ours is 7d41d4ce-14a6-11dc-8314-0800200c9a66

**Line 6:** this is the closing tag, you are telling the xml document that the item you are adding is now added

**Line 7:** this concludes your item entry, as XML documents can contain multiple items, your are telling the document to end by pluralizing the argument and adding the closing tag of a “/”

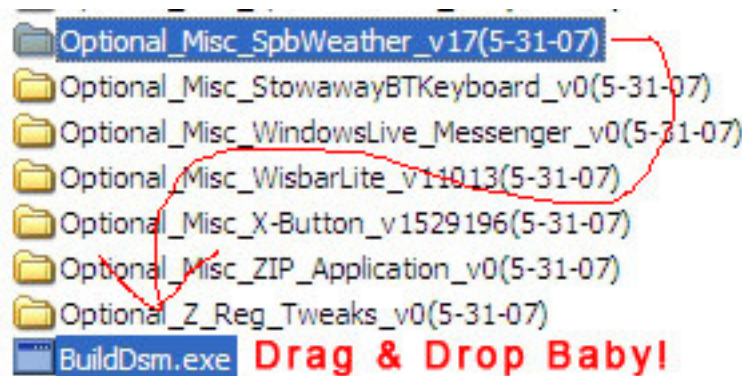
12 – Congratulations! The Final step (I promise) is to move your entire folder into the kitchens “TOOLS” folder. Good, now drag it on top of the file “BuildDsm.exe” ... wait a few seconds and a black box will popup, hit any key and it disappears, now move that folder into the OEM folder. YAY!

```

C:\Documents and Settings\jpl2441\Desktop\BigJ's_Reloaded_Kitchen\OEM\BuildDsm.exe
Bart.tsk
blankbanner.96.gif
Bliss.tsk
complete.96.gif
Default.wav
Default_stwater.gif
Default_stwater_240_320.gif
Default_stwater_320_240.gif
dlrback_land.png
dlrback_port.png
dlrbtndef_up_land.gif
dlrbtndef_up_port.gif
gves.exe.0409.mui
Mario_Oldskool.tsk
Matrix_Code.tsk
myinfo.96.gif
Organic.tsk
pegcards.dll
Planet_Moon.tsk
PurpleSunset.tsk
welcomehead.96.png
Windows_Default.wav
Worms_3D.tsk
Zerg_Planet.tsk
Press any key to continue . . .

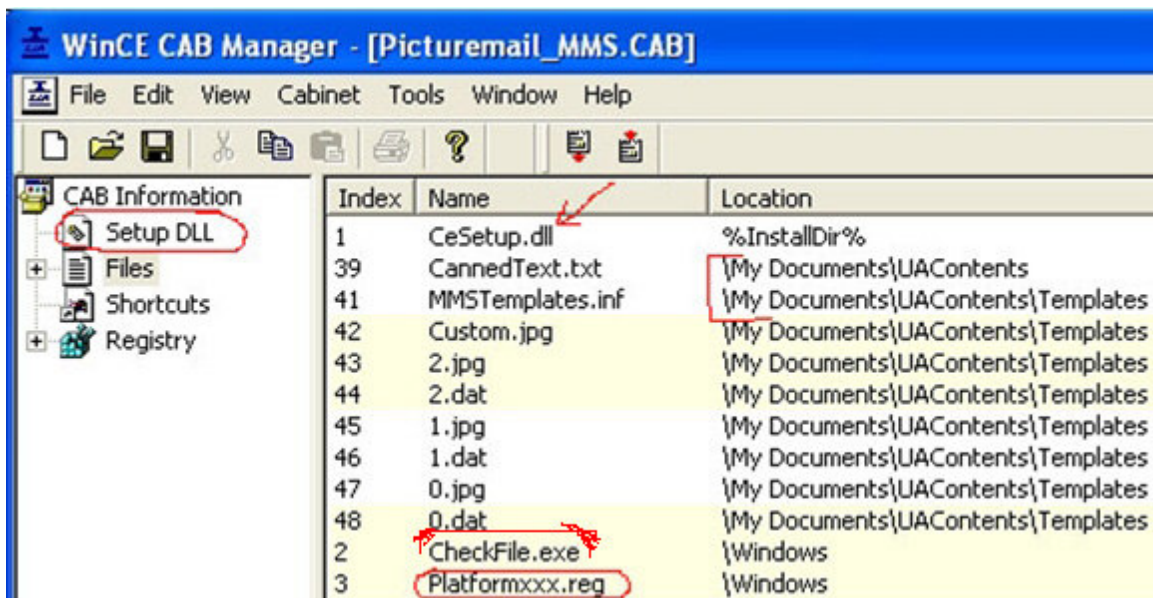
```

If you're clever you saw that I actually move BuildDsm.exe from TOOLS and into my OEM folder, that way I don't have to move the file all over, I just put it in OEM, and then drag it to BuildDsm.exe. Yes that file makes it into my phone, but worth the time I save. It is extremely important that you do not forget this FINAL step. If you forget to do this, then your package has a high chance of not being integrated properly, corrupted, or worse, it can actually break your package entirely, causing you to more or less start from scratch.



## Complicated OEM Packages

Complicated OEM packages are exactly that, complicated. These are packages that use tricks and shortcuts to accomplish the task of installing. Figuring out exactly what they are doing is a serious pain in the ass. I'm going to go into a few of the tricks employed, and ways to recognize them, and avoid missing something that is crucial to your OEM packages success. For this Example We are going to break down the Arcsoft Picturemail MMS cab file. This particular cab file is courtesy of the Verizon Extended Rom. I choose this one in particular because it pretty much uses every dirty trick in the book. One of which I am still learning to overcome, provxml files, so I will not be able to get into much detail once we get to that point.



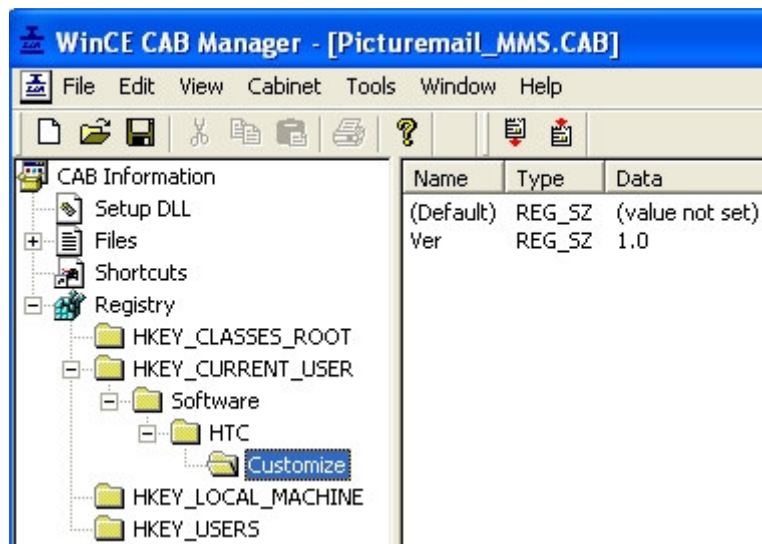
In this chapter we are going to cover

- Platformxxx.reg files & why they are a pain in the ass
- Setup.dll files & how they are devious little monsters
- Compressed or mis-named files
- Files that need to be put into sub-directories & how to move them
- How to determine if provxml files are needed and how little I know about them



*Platformxxx.reg*

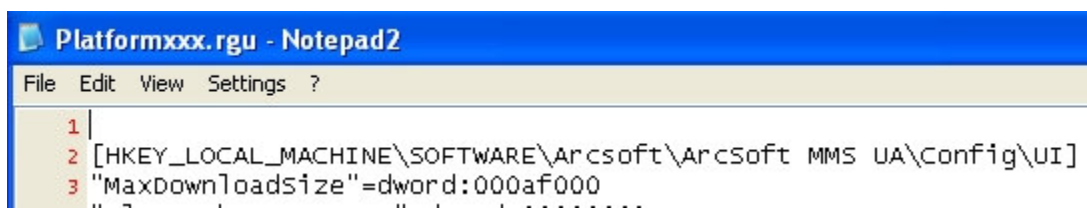
This one is really easy to spot simply because it's a file buried in the cab itself. You see it circled in the picture on the previous page. Whenever you see a file like this, buckle up baby cuz its about to be a long ride. These files are evil because they are typically very long making it a tedious process to go through it and correct all of the inconsistencies so that BuildOS can read it. If you miss the fact that there is a Platformxxx.reg, another way to know that the CAB file is using one is to look at the registry entries currently listed by WinCE Cab Manager.



Whenever you see this entry, it's a tell that you overlooked a Platformxxx.reg file. This entry tells the operating system, that the cab file has supplied a registry file that needs to be integrated into the registry (the Platformxxx.reg).

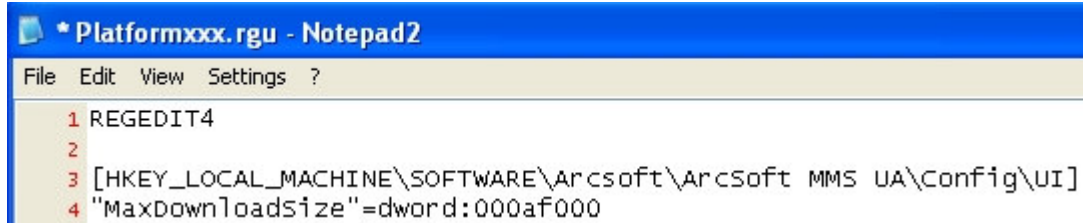
Extract all of the files as you normally would as we discussed in the previous example. Once this is complete you will need to follow basically the same steps as before, rename the registry file to .RGU, convert it to Unicode, make sure there is an empty line at the end, and double check that it starts with REGEDIT4 (some platformxxx.reg files do not start with REGEDIT4 and ALL rgu files must).

Now starts the fun part, remember that line-by-line interrogation that we talked about on big ass registry files, let's get started, look at the below example, what is missing?





The first line must always be.... REGEDIT4, so let's add this and move on.



```
Platformxxx.rgu - Notepad2
File Edit View Settings ?
1 REGEDIT4
2
3 [HKEY_LOCAL_MACHINE\SOFTWARE\Arcsoft\ArcSoft MMS UA\Config\UI]
4 "MaxDownloadSize"=dword:000af000
```

Now we can officially begin the search, here's a clip about 130 lines into this mess

```
131 [HKEY_LOCAL_MACHINE\SOFTWARE\Arcsoft\ArcSoft MMS UA\Config\mm1]
132 "SendDelayedNotifyResp"=dword:00000000
133 "Enabledrmf1"=dword:00000000
134 "WAP2DefaultPort"=dword:00000050
135 "WAP1DefaultPort"=dword:000023f1
136 "LastAckTimeout"=dword:0000fa0
137 "DebugPath"="\My Documents\UAContents\MMS Log\"
138 "EnabledDebug"=dword:00000003
139 "SuppressDelayedNotifyResp"=dword:00000000
140 "1"="6654"
141 "TotalSettings"=dword:00000001
142 "DefaultSetting"=dword:00000001
143 "WTPMaxRetryTimes"=dword:00000008
144 "SendAndRecvTimeout"=dword:00001388
145 "ReportAllowed"=dword:00000001
146 "EnableMSISDN"=dword:00000000
147 "WAPDefaultTimeout"=dword:00001388
148 "HTTPDefaultTimeout"=dword:000493e0
```

Look carefully, at first glance this looks like a legitimate section of registry.

```
137 "DebugPath"="\My Documents\UAContents\MMS Log\"
```

Line 137 has pathing issues. As we discussed earlier all entries that are built need to have pathing done with double "\\" So let's correct this.

```
137 "DebugPath"="\My Documents\\UAContents\\MMS Log\"
```

This looks right now doesn't it? NO! You made a rookie mistake, well I never told you so I'll have to let it slide this time, Registry pathing in an RGU file cannot end in "\\" because it thinks there is more to the path and will throw an error when building your ROM when it hits this OEM Package. Let's make the final correction here and move on.

```
137 "DebugPath"="\My Documents\\UAContents\MMS Log"
```

Great, only 190 or so more lines to go through. Ya, Platformxxx.reg files suck balls. This particular one has about 60 or so corrections like the above to make.

## Setup DLL files

When you open a CAB file and see right at the top “Setup DLL” you know you’re in for a multiple flash session to test this OEM. This clues you into a number of things going on with the cab file in question, all of which you will not like.

First, there is probably a “CeSetup.dll” file, and/or SetupDll.dll, and/or SettingDll.dll

Second, there is sometimes going to be a “Checkfile.exe”

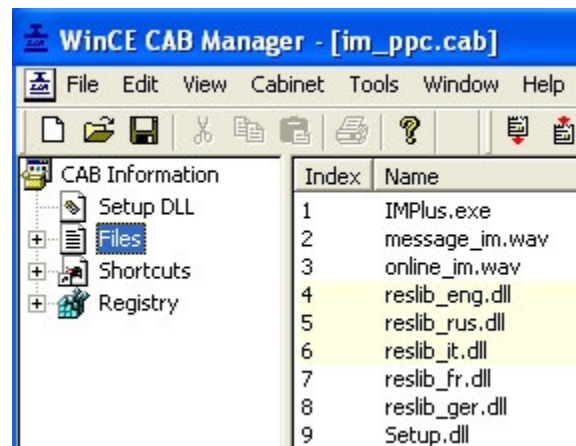
Third, some of the files in the package *may or may not* be renamed during the install

Fourth, additional registry entries *may or may not* be created after the install

First & Second are easily taken care of, just delete those files outright from your extraction folder -which should be named..... Optional\_MMS\_v36124(5-31-07)

You want to delete those files because they do nothing for an OEM integration, and since a lot of other cab files will be using those exact same files, you will get duplicate file errors when running BuildOS.

The third item is the mystery item. You typically won’t know this until you build the package, burn a ROM, and then throw some random error about missing a file. An example of this is the IM+ Cab file.



Here you will see a nice list of “reslib\_” language files. When you install the application it asks you what language to install, when you choose, it takes the associated language file, names it resource\_lang.dll and then copies it into the windows directory *not* the programs directory. The only way to figure this out, is to build the whole package and then try to launch the program and get thrown a “resource\_lang.dll file is missing or corrupted” error. Then you get to revisit the CAB file and try to figure out which one of those files is being renamed, and then you realize you just need to rename one, and delete the rest from your package because they are not used. Did you notice the Setup DLL at the top of that cab file.... I did, see the Setup.dll file in there to that needs to be deleted?.... EVIL!

The fourth thing a Setup DLL file can do, is generate registry entries that the program needs to function. Sometimes the CAB will have registry entries, and yet, the Setup DLL will add even more additional entries that were not exported with the rest. The only way to get these entries is to install the cab file on your PPC, and search the registry. Sometimes it's simple, and you'll just find more entries underneath the software's heading, sometimes it's a pain in the butt, and you'll find them buried under a CLSID entry, if you can't find them, but know they are there, you will need to dump the whole registry, install the application, and redump the registry and run a comparison. Registry comparisons are the absolute worst thing you can ever sit through, it's a veritable hell of entries that make no sense. The ppc writes sooooo many erroneous things to the registry that doing a full comparison can take you hours. Those are the applications that I typically give up on if I can't find the entries I need within an hour or so.

When it's simple, just use resco registry, or its free equivalent, and export the keys you need. Once on your computer, open them up and append them to the OEM's RGU file.

### ***Compressed or mis-named files***

Every once in a great while you will come across a cab file that has compressed files inside of it. I've even come across a few that have cab files *within* cab files. Those are kind of messy, but now that you know all of the steps, you just duplicate them, but all inside of the same OEM package.

You can recognize a compressed file easily, it will have an underscore where the last letter of the file is supposed to be. Instead of

Playback.dll – you will see

Playback.dl\_

The underscore denotes compression. The funny thing is, if you just rename it, it seems to work about 9 times out of 10. The other time, you need to install the cab file, and then go into that programs directory and copy that file out and get it onto your computer and into your OEM package folder. Simple, yet annoying all at the same time.

### ***Creating sub-directories and moving files into them***

For this section we are going to revisit our shortcut placement file. Remember way back when we were making shortcuts and had to tell the system where to actually *put* that shortcut? That's right, "initflashfiles.txt" Now pay close attention because this gets messy pretty quickly.

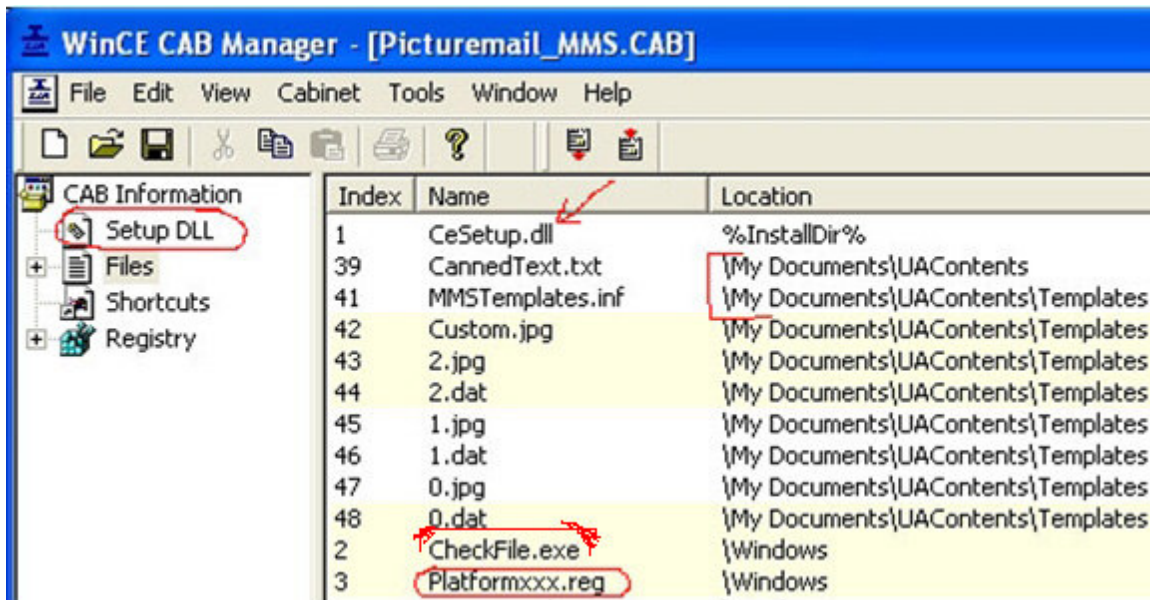
First of all you need to know if you need to create subdirectories at all

Secondly, you need to know if those directories already exist on the phone

Third, move all of the files you need to into their respective subdirectories

Fourth, you *CAN* rename the files when you move them (awesome I know)

#1 – Detecting if you need to move files into subdirectories is fairly straightforward. Lets look back on this picture example of a complicated OEM package shall we.



As you can see, there are two directories that require a load of files to be moved into them. In this example we need a “UAContents” directory as well as a sub directory of that “Templates”. Lets go ahead and make ourselves an initflashfiles.txt and get started. Remember to comment what you are doing.

```
=====
; File Setup for ArcSoft MMS CDMA v3_0_6_24
=====
```

This is a good start for the top. Now before we get crazy, lets move all of the files that need to be moved into directories the handset already has. The phone already has a “My Documents” folder, it also already has a subdirectory of that called “My Music”

```
; -----File Movement into EXISTING directories-----
Directory("\My Documents\My Music");-File("Alouette.mid", "\Windows\Alouette.mid")
Directory("\My Documents\My Music");-File("ding.amr", "\Windows\ding.amr")
```

Now were going to CREATE the new directories that we need under “My Documents”

```
; -----Creating new directories on the handset-----
Directory("\My Documents");-Directory("UAContents")
Directory("\My Documents\UAContents");-Directory("Templates")
```

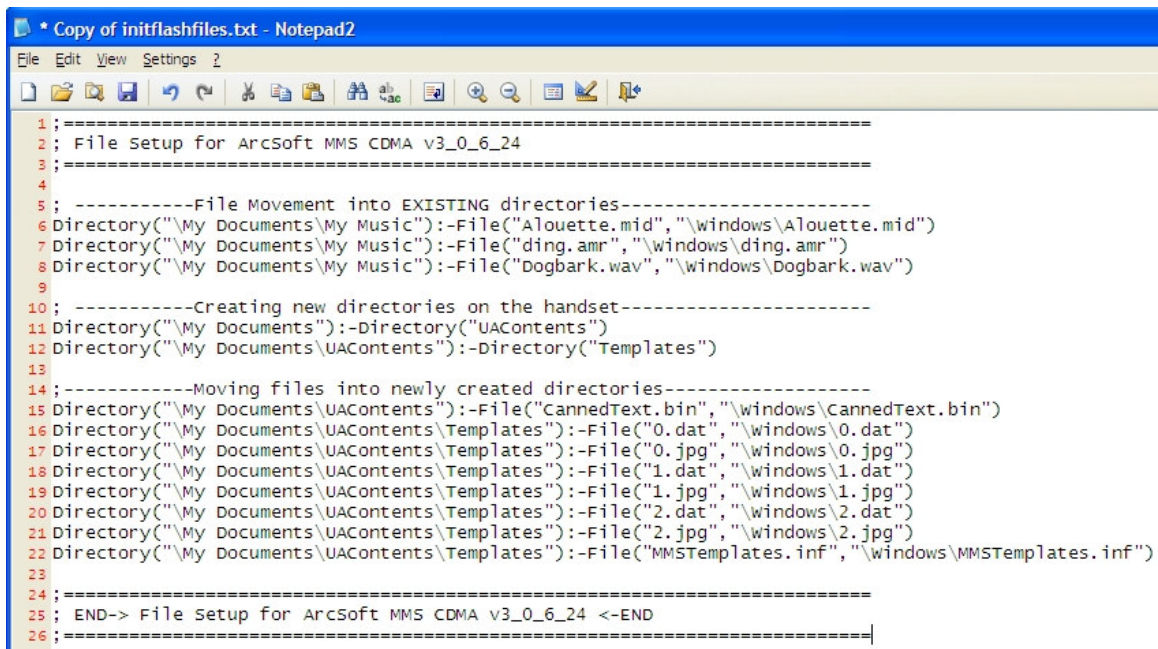
Once these directories are created we are free to move files into them

```
Directory("\My Documents\UAContents\Templates):-File("0.dat", "\Windows\0.dat")
Directory("\My Documents\UAContents\Templates):-File("0.jpg", "\Windows\0.jpg")
Directory("\My Documents\UAContents\Templates):-File("1.dat", "\Windows\1.dat")
Directory("\My Documents\UAContents\Templates):-File("1.jpg", "\Windows\1.jpg")
Directory("\My Documents\UAContents\Templates):-File("2.dat", "\Windows\2.dat")
Directory("\My Documents\UAContents\Templates):-File("2.jpg", "\Windows\2.jpg")
```

Now of course we need to close the file out properly, like so...

```
=====
; END-> File Setup for ArcSoft MMS CDMA v3_0_6_24 <-END
=====
```

I cropped a few entries out because I think you get the point, here is the file now in it's entirety



```
1 ; =====
2 ; File Setup for ArcSoft MMS CDMA v3_0_6_24
3 ; =====
4
5 ; -----File Movement into EXISTING directories-----
6 Directory("\My Documents\My Music):-File("Alouette.mid", "\windows\Alouette.mid")
7 Directory("\My Documents\My Music):-File("ding.amr", "\windows\ding.amr")
8 Directory("\My Documents\My Music):-File("Dogbark.wav", "\windows\Dogbark.wav")
9
10 ; -----Creating new directories on the handset-----
11 Directory("\My Documents):-Directory("UAContents")
12 Directory("\My Documents\UAContents):-Directory("Templates")
13
14 ; -----Moving files into newly created directories-----
15 Directory("\My Documents\UAContents):-File("CannedText.bin", "\windows\CannedText.bin")
16 Directory("\My Documents\UAContents\Templates):-File("0.dat", "\windows\0.dat")
17 Directory("\My Documents\UAContents\Templates):-File("0.jpg", "\windows\0.jpg")
18 Directory("\My Documents\UAContents\Templates):-File("1.dat", "\windows\1.dat")
19 Directory("\My Documents\UAContents\Templates):-File("1.jpg", "\windows\1.jpg")
20 Directory("\My Documents\UAContents\Templates):-File("2.dat", "\windows\2.dat")
21 Directory("\My Documents\UAContents\Templates):-File("2.jpg", "\windows\2.jpg")
22 Directory("\My Documents\UAContents\Templates):-File("MMSTemplates.inf", "\windows\MMSTemplates.inf")
23
24 ; =====
25 ; END-> File Setup for ArcSoft MMS CDMA v3_0_6_24 <-END
26 ; =====
```

Now lets imagine that you go to BuildOS.exe and you get a file error, that there is already files named "0.dat" "1.dat" & "2.dat" What do you do now? This is where the magic comes in. You can rename files on the fly with initflashfiles.txt so long as they are going into different directories. This is a great trick as well, to stay organized.

Since 0, 1, & 2 are all duplicate files, we will rename them in our OEM package to 1mms.dat, 2mms.dat and 3mms.dat





Fantastic, now we need to change our initflashfiles.txt slightly to compensate for this change. Open it up, on the lines where we moved 0 1 2 dat files, were going to make alterations

```
Directory("\My Documents\UAContents\Templates");-File("0.dat","\Windows\0mms.dat")
Directory("\My Documents\UAContents\Templates");-File("0.jpg","\Windows\0.jpg")
Directory("\My Documents\UAContents\Templates");-File("1.dat","\Windows\1mms.dat")
Directory("\My Documents\UAContents\Templates");-File("1.jpg","\Windows\1.jpg")
Directory("\My Documents\UAContents\Templates");-File("2.dat","\Windows\2mms.dat")
Directory("\My Documents\UAContents\Templates");-File("2.jpg","\Windows\2.jpg")
```

Congratulations, you have just renamed 0 1 & 2 dat files back to their original names. You see, the second filename is that which you have in your OEM package. The first name is what the file will be named once it is moved to its destination.

This works for shortcuts as well. Lets revisit our “DeviceLock.Ink” shortcut file. Who really wants a shortcut called “DeviceLock” on their phone? You don’t, so were going to rename that file to “Device Lock” once it’s made as a shortcut. Here is the original example

```
1 ; =====
2 ; Device Lock v1.2 shortcut Placement
3 ; =====
4 ;
5 Directory("\windows\Start Menu\Programs");-File("DeviceLock.Ink","\windows\DeviceLock.Ink")
6 ;
7 ;
8 ; END-> Device Lock v1.2 Shortcut Placement <-END
9 ; =====
```

And again with the shortcut being renamed, effectively adding a space to the shortcut

```
1 ; =====
2 ; Device Lock v1.2 shortcut Placement
3 ; =====
4 ;
5 Directory("\windows\Start Menu\Programs");-File("Device Lock.Ink","\windows\DeviceLock.Ink")
6 ;
7 ;
8 ; END-> Device Lock v1.2 Shortcut Placement <-END
9 ; =====
```



```
Directory("\\Windows\\Start Menu\\Programs"):-File("Contacts Backup.lnk", "\\Windows\\PPCBckpContacts.lnk")
Directory("\\Windows\\Start Menu\\Programs\\Games"):-File("Missile Defense.lnk", "\\Windows\\mcommand.lnk")
Directory("\\Windows\\Start Menu\\Programs\\Games"):-File("Texas Holdem.lnk", "\\Windows\\Holdem.lnk")
```

The screenshot shows a Notepad2 window titled "initflashfiles.txt - Notepad2". The menu bar includes File, Edit, View, Settings, and a question mark. The toolbar contains icons for file operations and editing. The text area displays the following script with red annotations:

```

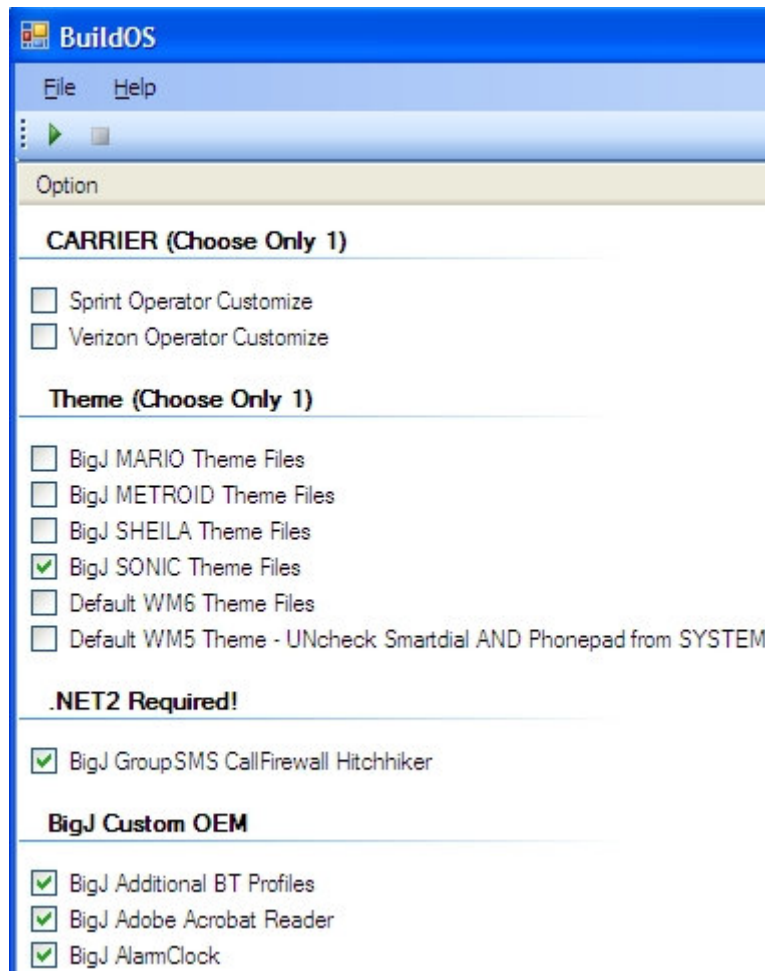
1 ;=====
2 ; START => EXAMPLE INITFLASHFILES.TXT <= START
3 ;=====
4 Destination Folder for files you
5 are moving -----STARTS UP WITH WINDOWS-----
6 Directory("%\windows\Startup"):-File("Batti.lnk", "%\windows\Batti.lnk")
7 Directory("%\windows\Startup"):-File("HandySwitcher.lnk", "%\windows\HandySwitcher.lnk")
8
9 ;-----SHORTCUTS IN PROGRAMS-----
10 Directory("%\windows\Start Menu\Programs"):-File("AlarmClock.lnk", "%\windows\AlarmClock.lnk")
11 Directory("%\windows\Start Menu\Programs"):-File("BT-Audio.lnk", "%\windows\BT-Audio.lnk")
12
13 ;-----GAME SHORTCUTS-----
14 Directory("%\windows\Start Menu\Programs\Games"):-File("Cribbage.lnk", "%\windows\Cribbage.lnk")
15 Directory("%\windows\Start Menu\Programs\Games"):-File("Chess.lnk", "%\windows\Chess.lnk")
16
17 Creating sub-directory under "My Documents"
18 ;-----CREATING DIRECTORIES-----
19 Directory("%\My Documents"):-Directory("%\UAContents")
20 Directory("%\My Documents\UAContents"):-Directory("%\Templates")
21
22 ;=====
23 ; END => EXAMPLE INITFLASHFILES.TXT <= END
24 ;=====
  
```

Annotations with red lines pointing to specific parts of the script:

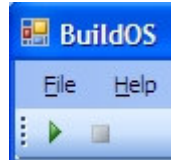
- Opening Comment Line:** Points to lines 1 and 2.
- Destination Folder for files you are moving:** Points to line 4.
- STARTS UP WITH WINDOWS:** Points to line 5.
- Target File name:** Points to the second argument in line 6.
- Source File from OEM:** Points to the second argument in line 11.
- Secondary Comment Line:** Points to line 12.
- CREATING DIRECTORIES:** Points to line 18.
- Creating another subdir under the previous one:** Points to line 19.
- End Comment Line:** Points to lines 23 and 24.

## Section III: The ROM Kitchen

A lot more complicated than you thought. Now comes the easy part believe it or not. Now we are going to go through, step-by-step on using the kitchen to burn yourself a customized ROM. Take a deep breath and launch BuildOS.exe

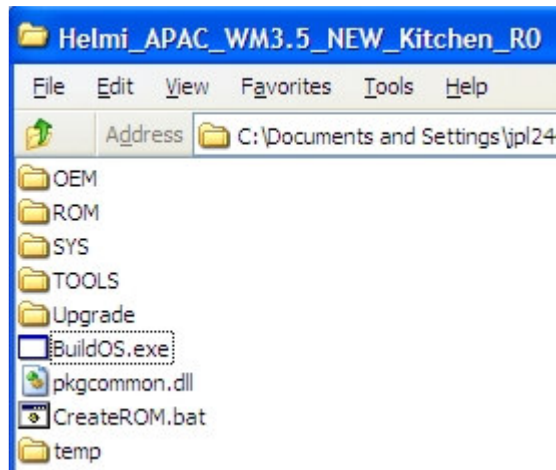


If everything was done properly, it should open with no errors and present you with the list of categories and packages that you setup in the OEM folder. If you receive an error, Go to section IV for troubleshooting and then return here. You can see here that I labeled some categories to "Choose Only 1". This is because choosing more than one will build conflicting files and/or registry entries, for instance, you can only have ONE crossbow theme, to do more than one will throw errors for file duplication. Now is a good time to double check to make sure EVERY SINGLE OEM package you put in the OEM folder is showing up. If not, revisit the option.xml section and return here. Now click the build button and sit tight. It's the little green one at the top, the application will gray out and in the bottom right corner you will see its progress, on the bottom left you'll see the current folder it's processing.



Processing: SYS\APACHE\_DRIVERS

Once its completed, you will see the word Done! In the bottom left and the screen will return to white. You will also see a new directory created in the root of your kitchen called “temp”.



Now we will create the ROM you will flash to your handset. Close BuildOS and double click on “CreateROM.bat” It will hang for about 20 seconds at this point and finally bring you to this prompt

```
C:\ C:\WINNT\system32\cmd.exe
.\rom\nk.nba
1 File(s) copied
Searching for IMGFS start... Found at 00640000
Dumping IMGFS ...
Done!
Next: Build the rom...
Press any key to continue . . .
```

Look at your keyboard and find the “ANY” key. Press the “any” key and it will then package up your dump folder (/temp/dump/) into an NBA file. This happens at a breakneck speed so watch out for whiplash the below screenshot was slowed down so we could capture a glimpse of what is taking place...

```

C:\WINNT\system32\cmd.exe
Processing "indialer_zoomsel_1.bmp" as file
Processing "indialer_zoomsel_p.bmp" as file
Processing "indialer_zoom_1.bmp" as file
Processing "indialer_zoom_p.bmp" as file
Processing "infantry.dll" as module
Processing "Infbeg.wav" as file
Processing "Infend.wav" as file
Processing "initflashfiles.dat" as file
Processing "InitProv.dll" as file
Processing "inkobj.dll" as module
Processing "inkx.dll" as module
Processing "input.2bp" as file
Processing "input_kboard.bmp" as file
Processing "inreplce.dll" as module
Processing "IntelliDialer.htm" as file
Processing "intshare.dll" as module
Processing "IntShrUI.exe" as module
Processing "IntShrUI.lnk" as file
Processing "iphlpapi.dll" as module
Processing "ipnat.dll" as file
Processing "ipsec.dll" as module
Processing "ipsecsvc.dll" as module
Processing "ipv6hlp.dll" as module
Processing "ircomm.dll" as module

```

WOW! That was quick huh? This will finish up in a matter of seconds and bring you to the next prompt

```

Total Sectors: 00000
Used Sectors : 12cc1
Free Sectors : fffed33f
Searching for IMGFS start... Found at 00640000
Fixing... Done!
Next: Flash the rom...
Set the device to bootloader mode if necesarry...

!!!WARNING!!!!
Press any key to continue . . .

```

This bat file really likes the “any” key. Go ahead and hit the “any” key to continue.

```

!!!WARNING!!!!
Press any key to continue . . .
1 file(s) copied.
1 file(s) copied.
1 file(s) copied.
1 file(s) copied.
1 file(s) copied.
1 file(s) copied.
1 file(s) copied.
1 file(s) copied.
ECHO is off.
Select encode and select your operator name on the next dialog to avoid country
error...

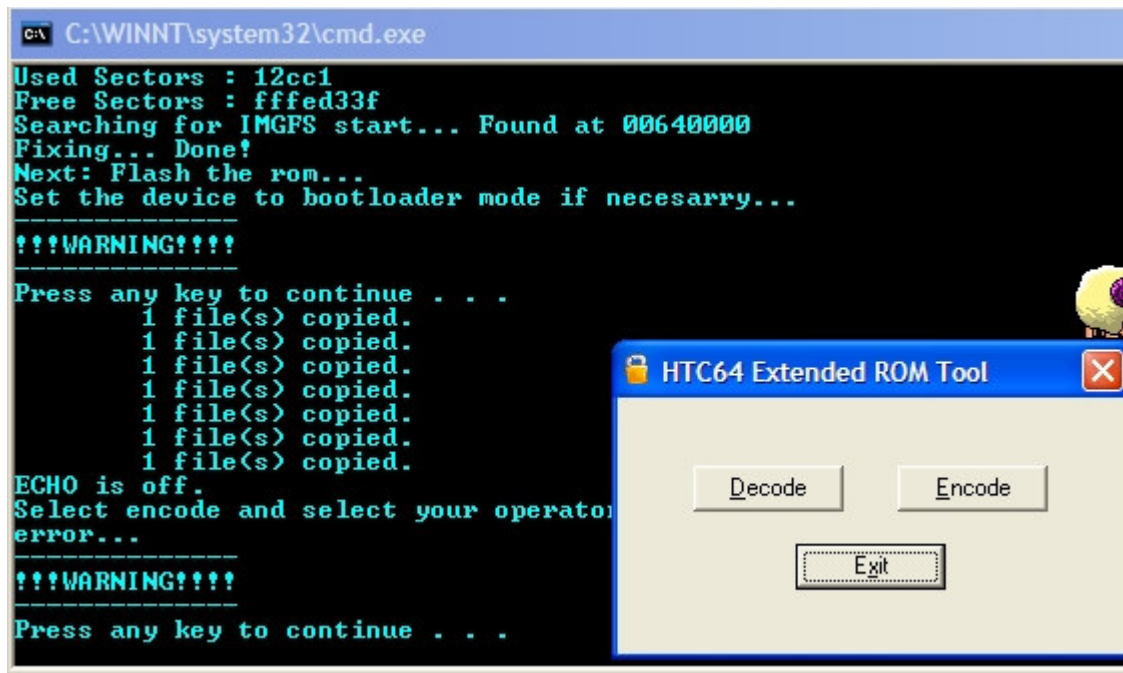
!!!WARNING!!!!
Press any key to continue . . .

```

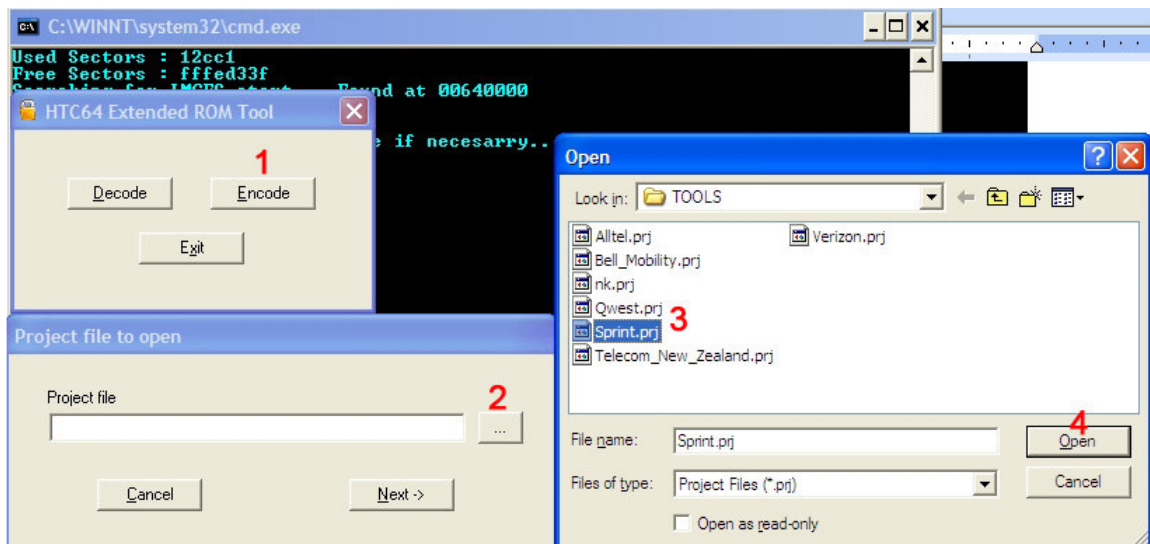
Hmmmm, that yellow text there looks important. This is a HUGE issue on the forums, learn how to read guys, if you don’t choose the right carrier you will have issues.



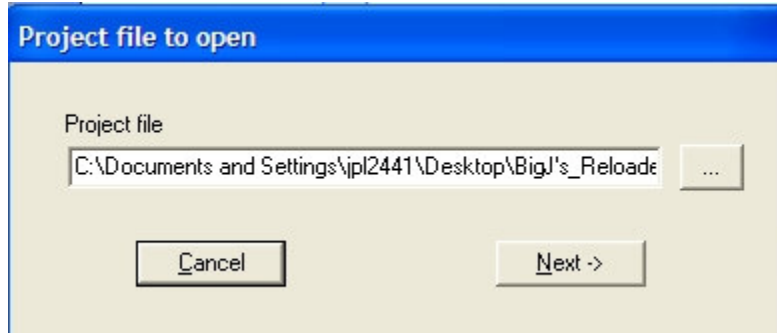
Hitting your “any” key now will launch the encode application. This allows you to encode your ROM using the **PROPER** carrier file.



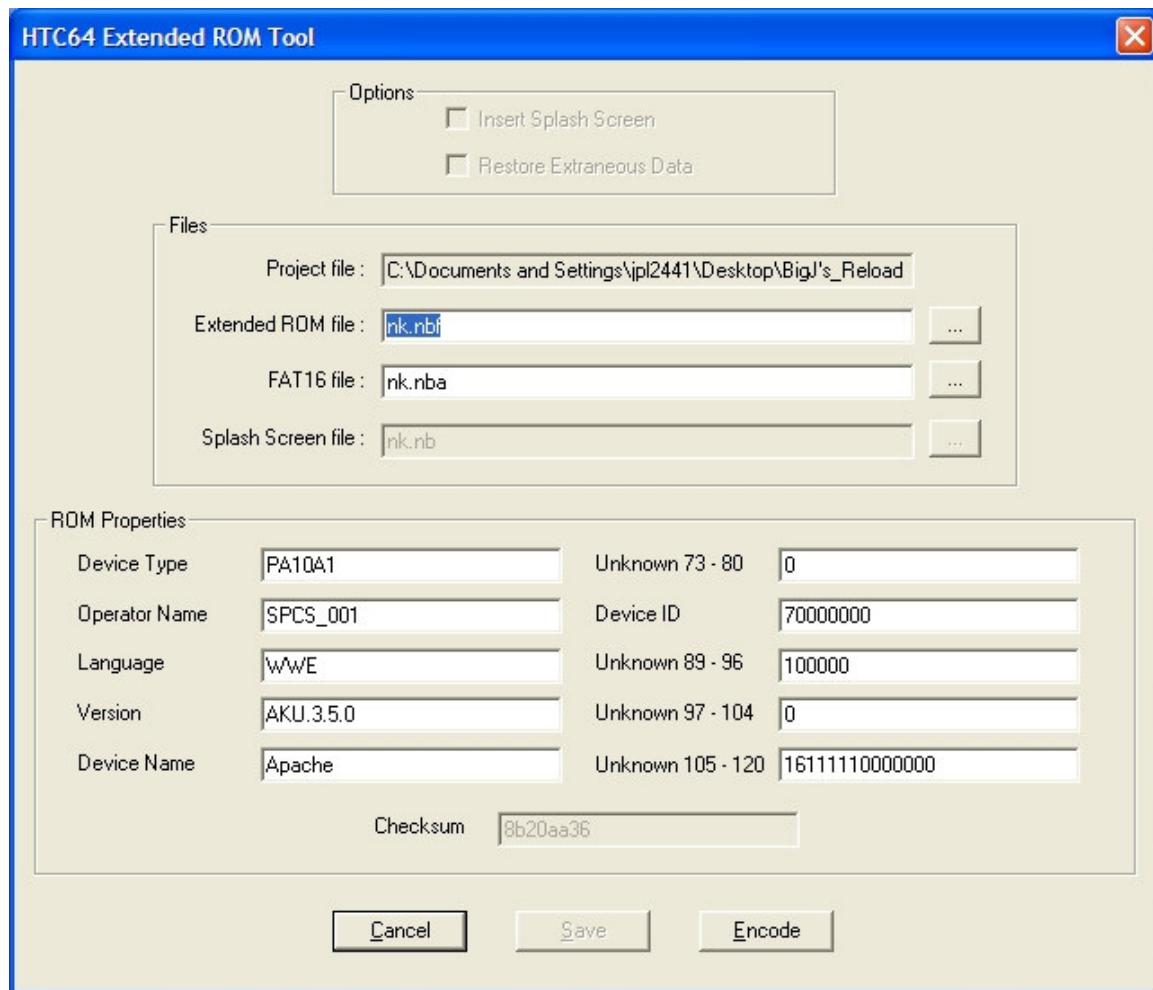
Now click on Encode, choose the three “...” to locate the encode file, choose your carriers file, and finally click on open. A flowchart has been provided.



The project file path should now be populated.



Click on Next-



This is where the magic happens. This program will encode your nba file into a fully flashable nk.nbf file. The **ONLY** thing you can modify here is the “Version” Changing anything else will fubar your ROM and you will need to start over. I wouldn’t even mess with the Version tag for right now, not till you have made a successful ROM. Click

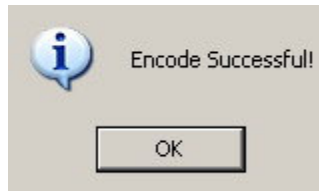
Encode



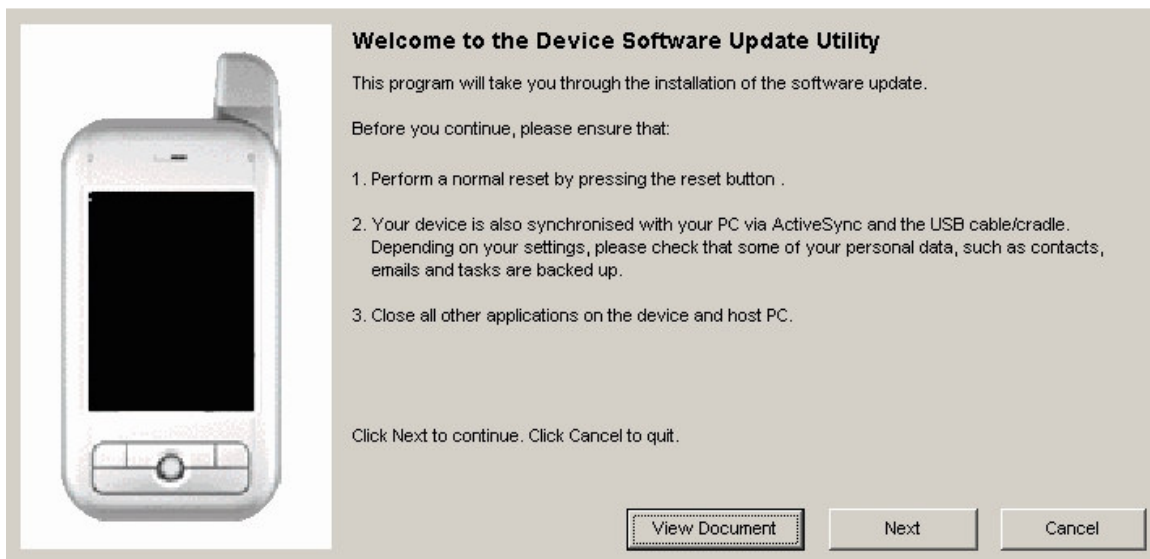
At this point you *\*may\** get an error that looks like this.



Just click OK and it will encode

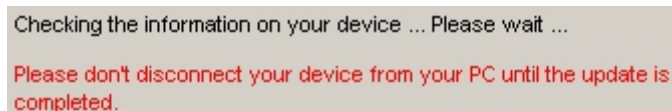


Once it encodes successfully it will dump you back to the black screen with the command prompt. Once again find your "any" key and hit it. This will open up the phones flash utility and allow you to begin flashing your newly made ROM to your handset.



Click on Next. This will then "detect" that your phone is hooked up to the computer and allow you to proceed with your flash. Remember to

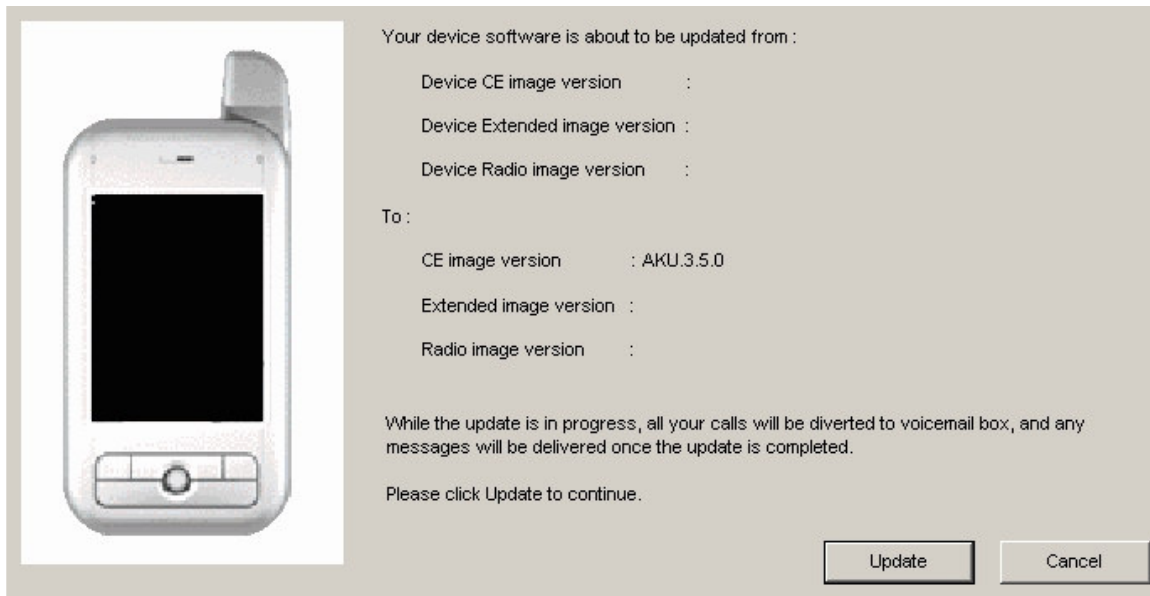
***BACK UP YOUR DATA BEFORE FLASHING!!!***



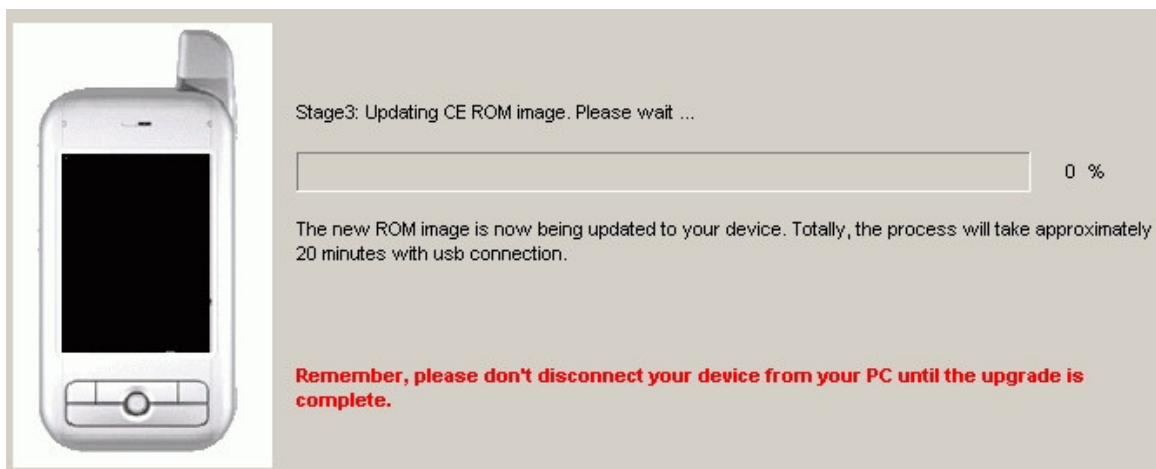
I see you skipped that last part so let's re-iterate that one more time

***BACK UP YOUR DATA BEFORE FLASHING!!!***

Now we are at the final screen before wiping your data, and updating your OS.



Click Update. Sit back, this takes approx 15 minutes. You can watch the little bars go up as it progresses, you can even count them like I do...



When it finishes it will bring you to a completed screen. Click on Finish and unplug.

***HARD RESET YOUR PHONE NOW!***

Hard-Reset your phone. Go through the setup and TEST every OEM package you included to make sure its working. Test the built-in applications as well. Double check that your carrier connection is there (#777), pretty much, TEST TEST TEST everything!

Congratulations, you're a PPCGeek!

## *BigJ's Tips & Tricks*

Here is a little segment on things I do differently. Some are done to speed things up. Others are done purely for organizational reasons, and others... well lets just not talk about those.

The very first thing I do is edit the CreateROM.bat file. I don't like having all that garbage automated, I want it to do one thing, make my dump folder an .NBA file. Thanks guys, but ill take it from here. To do this you right click on the CreateROM.bat file and choose EDIT. At the very first WARNING segment I delete everything and add in the :end of the file. This stops it from doing anything after making the nba, and just closes the program.

```
ECHO Next: Flash the rom...
ECHO Set the device to bootloader mode if necesarry...
ECHO -----
ECHO !!!WARNING!!!!
ECHO -----
pause
@ECHO OFF
CD ..\Tools\
COPY Telecom_New_Zealand.prj ..\temp\Telecom_New_Zealan
```

For me, this becomes...

```
ECHO Next: Grab the *.nba file from /temp/ and patch the bootscreen...
ECHO Then: convert the patched NBA file to nbf and flash!
ECHO -----
ECHO !!!WARNING!!!!
ECHO -----
@ECHO OFF
:end
pause
```

I put in two lines of notes to remind me to grab it and patch it, then encode it and burn it!

Doing it this way allows me to grab the raw .nba file from the /temp/ folder and then move it to my patch folder. This is where I patch in my custom boot screens. It also allows me to use my bigj.prj file to encode the nba file to an nbf, my personal encode file is where I have modified the Version key that we talked about leaving alone previously. After I encode it to an NBF file, I throw it into my patch folder. This is the one that contains the NOID exe that forces you to be in bootloader mode when flashing, and also ignores all country id type errors. I don't trust flashing a phone that is on, I prefer to do it in bootloader.

Most of my other tricks we have gone over. I have discussed cleaning up the kitchen to be better organized by clearing out the OEM folder for optional\_ item packages only. We discussed creating and cleaning up OEM packages. Every SINGLE oem package on the ftp I have used, I have taken, cleaned up, redone the initflashfiles.txt. We talked about making cute checkboxes for everything by using an option.xml file. So ya, take the initiative and clean up your kitchen. The single best advice I can give you is to TAKE NOTES. *Every Single Change....* And I mean every single thing I touch, move, open, delete.... I note down in notepad as I work. That way I don't forget something, or break something and not remember which files I was into.

Tricks we did not cover.

- 1 – UPX files executables and DLL files to get more bang for your buck
- 2 – Why I like to remove help files, and how totally worthless they are
- 3 – How I suck at provxml files but know enough to be dangerous

- 1- UPX files. This is a fantastic way to cram more stuff in your ROM. It basically “zips” up the exe file and can do the same for dll files. Some programs will not function properly when compressed, or when you compress their support files so be careful and thoroughly test those programs before making an OEM with UPX compressed files. There are numerous rumors that compressing the files wont allow you to gain more room in your ROM, its called a rumor for a reason, and its because that simply isn't true. I had hit the ceiling for space in my custom roms several times. The easiest way for me to cram another program in there was by compressing other programs that were already built in. Another thing you may hear, is that the file FULLY decompresses into RAM when you load it. This is only partially true. Doing some research I found that it is only executed fully into RAM initially, once its decompressed and running, the file itself leaves only a footprint in RAM that points to the compressed file. Its kind of hard to understand from a technical standpoint so let me simplify it a little. ~ BigJ's Metaphor ~ Its like leaving a trail of rice behind you as you walk into a maze. Uh, so feel free to read all the documentation on sourceforge if you really care.

- basically `mxip_APP_VERSION.provxml` pretty straightforward. You can also use `provxml` files to create and delete registry entries, but its not necessary 99% of the time so it was not worth making an entire section on.

.....



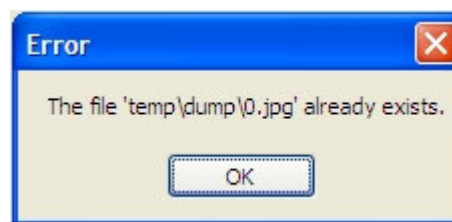
## *Section IV – What went wrong?*

In this section we are going to discuss all the different types of things that can go wrong. This will be as comprehensive as possible within reason, but it should be enough for you to overcome the most common problems associated with building your own ROM, and how to resolve them without bothering people like me...

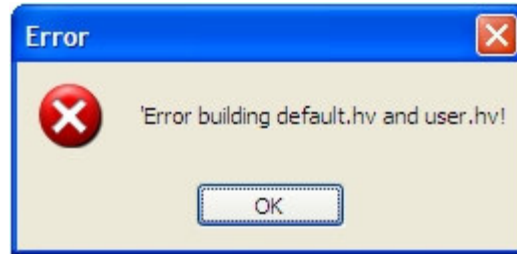
- Common Error messages and what they mean
- Stuck in Bootloader Mode
- White Screen of Death
- Hanging on the Bootscreen
- Hanging on the second splash screen
- Locks up at the today screen

The VERY first error you will encounter will be a mistake in your option.xml files. Either the one in the root directory of OEM or SYS, and secondly, one that you put into an OEM package. The program will open, and then just promptly crash asking you to send a crash report to Microsoft. This is an OPTION.XML conflict. To avoid this you'll want to launch BuildOS.exe after every OEM package that you add to the kitchen, as well as any time you make a change to either of the root option.xml files (which you really should never be touching anyways).

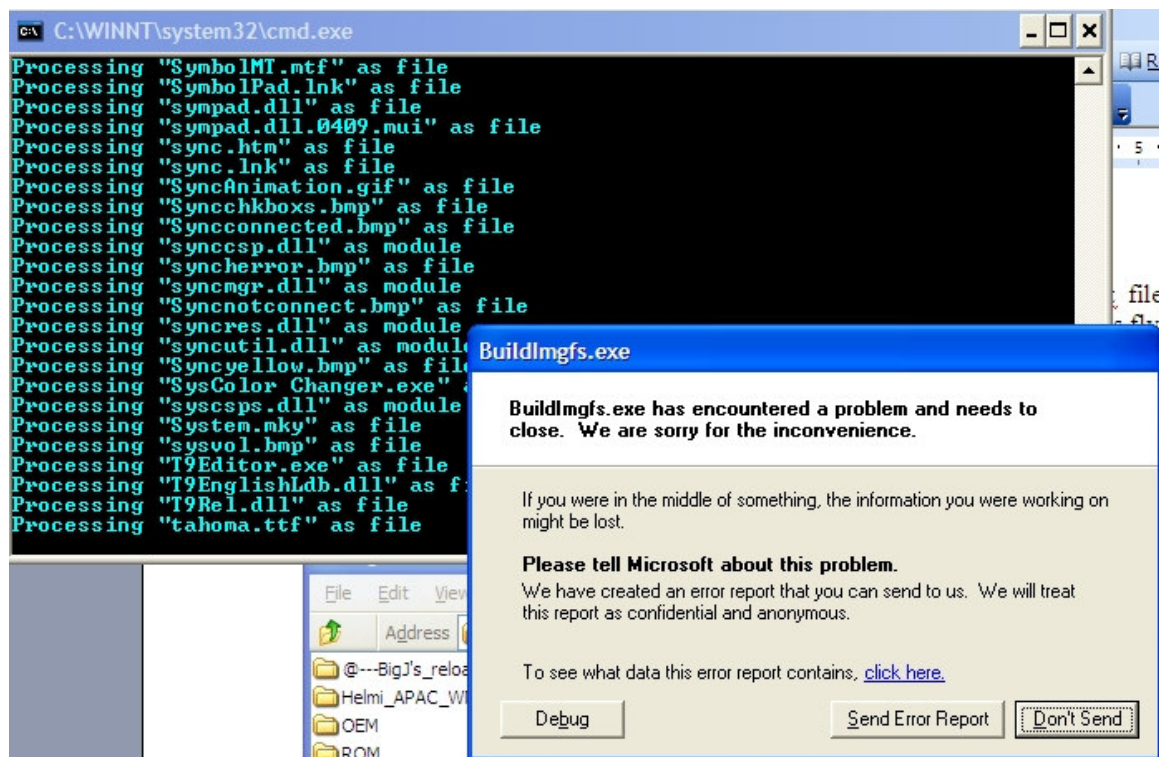
If you have built OEM packages that have files that are already in the ROM, you will get this error. Note the name of the file in question in the error dialog, and do a search for it in your kitchen folder. You will only need one instance of any given file. If you do in fact need both files, rename one, and edit its initflashfiles to change its name once it's moved to its final resting place. If this is not an option for you, you will need to decide which copy of the file must go.



Another error you will probably encounter is the one that BuildOS throws when you screwed up an RGU file. It's pretty nasty and you have to wait until the very end to get it. Basically it's attempting to create an entry that is not formatted properly. This can be caused by a few things. We discussed already about having file paths use double "\\" and not ending with them. You also need to double check that your .RGU files are actually encoded in UNICODE. This is very important if you remember.



The next error you may get is actually running the CreateROM.bat file. The initial portion will run properly, once you hit the any key, and all the text goes flying down the list you will get an error. The funny thing is, it lets you continue to build the ROM, but you should not, it won't work anyways. The error you see below, means that you overfilled your ROM. That's right, you put WAY to many OEM packages into your build. You need to go through some elimination tactics and put less packages into your ROM.



The next three errors are not an issue with this kitchen. The previous kitchens would not error when you went overlimit, and as such, you would build up the ROM and never know you were over size. Since this is no longer an issue, I will not be going into detail on them. If this happens to you for some reason at this point, it's safe to say that you probably messed up something in the SYS folder, or one of your OEM packages overwrote or bypassed something in the registry it should not have. They are...

- White Screen of Death
- Hanging on the Bootscreen
- Hanging on the second splash screen

The last issue we are going to discuss is hanging on the todayscreen. Typically you boot up your device, there is a slight delay, and then all of the today screen plugins load. When it doesn't, and the phone is actually hung on that initial boot phase while loading the plugins... guess what... you have a plugin problem.

I personally disable most of the operating system plugins by editing their boot string to zero in the rgu file. (see my registry tweaks OEM). A lot of OEM packages set their programs to load on the today screen by default. This is great if you're installing the program, but were not installing the program, we have made it an integral part of the operating system. Best practices say that *\*ANY\** non operating system plugin be disabled by default. You should follow that rule also. Sure it's neat to load up and have all your favorite plugins up right away, but they sometimes cause problems *especially* on an initial boot from a flash. It would be a good idea to change their load strings to zero, and after you have gotten into the system, go into the today-screen options and enable them. This will also help you know which application may have been causing the program because if you load it, and it crashes the phone, you need to revisit that OEM package.

I could go on forever, but 40 pages is my limit. I hope that this document is an invaluable tool for you, and for others as well. I would like to thank everyone in the PPCGeeks forums, as well as everyone on the XDA Developers boards. These two forums have never left me without an answer, and have provided me with all of the tools I have ever needed or wanted for my handset.

Thank you for looking at this guide, may it serve its purpose long after I have moved on.

~Fin