

[TUT] Sous-Chef's Guide to Da_G's Ervius Visual Kitchen

Version: 31/05/2009

Intro

Welcome; I wanted to offer a little “something” back to the XDA community in the hopes that will benefit others and to show my appreciation to the folks that make XDA the great community that it is. Hopefully, this guide will help you work your way up the ranks to Chef ... let's begin!

There are many chefs that provide quality ROM's for you to use. However, if you've gotten excited about the idea of cooking your own ROM's, you've probably felt overwhelmed by the volume of *Forum Threads* and *Wiki* pages at your disposal to learn how to do this.

The sections are intended to be followed in sequence as the last section should provide you with a final product that can be flashed to your device – so you may want to read this guide once over before going through the motions ... who am I kidding? You're going to follow along aren't you? ;)

In case you're wondering ... I chose *Sous-Chef* because *Commis* or *Chef De Partie* just didn't have the same appeal :)

Applying Original/Cooked ROM's

You probably won't be able to apply an Original or Cooked ROM to your device as your Cellular Carrier has most certainly locked your device. You'll need to unlock your device before venturing into the world of ROM installation. These activities are beyond the scope of this guide; you can however, go to this *Wiki* page to learn more.

[HTC Raphael](#)

http://wiki.xda-developers.com/index.php?pagename=HTC_Raphael

Original VS Cooked ROM's

HTC periodically releases Official Generic ROM's that you can apply to your device. You can find a list of Original Shipped WM6.1 ROM's at this *Wiki* page.

[Original Shipped WM6.1 ROMS](#)

http://wiki.xda-developers.com/index.php?pagename=HTC_Raphael_WM6.1_ROMs

There are essentially two types of Cooked ROM's; those that another Chef makes available for you to use, and those that you cook yourself. You can find a list of Available Cooked WM6.1 ROM's at this Wiki page.

Available Cooked WM6.1 ROMS

http://wiki.xda-developers.com/index.php?pagename=HTC_Raphael_Cooked_WM6.1_ROMs

Outro

Lastly, this guide only covers the ROM cooking process; changing your device **Startup Splash Screen** and **Radio** or flashing a **HardSPL** are beyond the scope of this guide; you can however, go to these *Wiki* and/or *Forum* pages to learn more.

Radio

http://wiki.xda-developers.com/index.php?pagename=Raphael_ExtractedRadioRoms

<http://forum.xda-developers.com/showthread.php?t=439566>

Startup Splash Screen

<http://forum.xda-developers.com/showthread.php?t=431161>

Hard SPL

http://wiki.xda-developers.com/index.php?pagename=Raphael_HardSPL

This guide is intended to help you learn how to cook your own ROM's; it will walk you through the process of extracting the contents of an Official ROM, adjusting the Page Pool, changing the Data Cache Size, and Patching the ROM to remove Certificate verification. The guide does not cover the steps required to add/remove ROM packages or port an XIP from a different ROM version or device ... not yet anyway ;)

Now for the disclaimer bit; I take no responsibility and will not be held liable for any problems you encounter with your device before and after following this guide ... flashing a ROM is done at your own risk. If you spot mistakes or inaccuracies in the guide however, please let me know so that I may correct them. Now, read on if you still feel that this is your choice career path ;)

Oh, one last thing ... special thanks to the following folks for sharing their knowledge with the rest of us ... thank you!

Da_G
Ameet
Bepe
Cmonex
Ervius
JCEspi2005
JugglerLKR
Olipro
Aruppenthal
NRGZ28

If I missed someone, it's purely accidental – send me a note and I will add your name to the list.

Location, Location, Location

There are many fine Kitchens out there to use; Semi-Automated Kitchens (Raphael, Da_G), Automated Kitchens (Bepe), and Visual Kitchens (Ervius). This guide uses the Ervius Visual Kitchen to assist you in learning the basics of operating a Kitchen; which ultimately, allows you to produce your own ROM.

References

Ervius
<http://forum.xda-developers.com/showthread.php?t=469420>

Da_G
<http://forum.xda-developers.com/showthread.php?t=471288>

Raphael
<http://forum.xda-developers.com/showthread.php?p=2453788>

Bepe
<http://forum.xda-developers.com/showthread.php?t=467488>

Inspecting the Facility

It's important to get acquainted with any new facility; last thing you want to find out is that you don't know where to plug your utensils or appliances. Here's a brief tour of the facilities to get you on your way.

EXT Folder

The External packages (EXT) folder is divided into **Device Specific** folders and a **Shared** folder. The **Shared** folder is typically used for files (packages) that can be installed on any device. **Device Specific** folders are typically used for files (packages) that should only be applied to a specific type of the device – for example, the Raphael device.

Each **Device Specific** folder is further divided into Operating System **Build Version Specific** folders and a **Common** folder. The **Common** folder is typically used for files (packages) that can be installed on any Operating System build version. Operating System **Build Version Specific** folders are typically used for files (packages) that should only be applied to a specific build (version) of Operating System – for example, build version 20764.

Operating System **Build Version Specific** folders, the **Common** folder, and the **Shared** folder can be further divided into sub-folders making file (package) management simpler – for example, `.\Raphael\20764\Raphael_External_Packages`.

Tip

- The **EXT Build** drop-down box in the Visual Kitchen permits selection of different Operating System **Build Version Specific** folders.

OEM Folder

The OEM folder can contain multiple **Device Specific** folders. **Device Specific** folders are typically used for files (packages) that should only be applied to a specific type of the device – for example, the Raphael device.

Each **Device Specific** folder is further divided into **Locale Specific** folders and a **Common** folder. The **Common** folder is typically used for files (packages) that can be installed in any locale – not specific to a language. **Locale Specific** folders are typically used for files (packages) that should only be applied to a specific locale – for example, 0409 (English).

Tip

- The **Language** drop-down box in the Visual Kitchen permits selection of different **Locale Specific** folders.

ROM Folder

The ROM folder is divided into **Device Specific** folders and a **Shared** folder. The **Shared** folder is typically used for kernel system files that are compatible amongst devices. **Device Specific** folders are typically used for kernel system files that should only be applied to a specific type of the device – for example, the Raphael device.

The **Shared** folder is further divided into Operating System **Build Version Specific** folders. Operating System **Build Version Specific** folders are typically used for kernel system files that should only be applied to a specific build (version) of Operating System – for example, build version 20764.

Each **Device Specific** folder is further divided into Operating System **Build Version Specific** folders. Operating System **Build Version Specific** folders are typically used for kernel system files that should only be applied to a specific build (version) of Operating System – for example, build version 20764.

Tip

- The **XIP Build** drop-down box in the Visual Kitchen permits selection of different Operating System **Build Version Specific** folders.
- Advanced *OEMXipKernel* and *MSXipKernel* operations can be performed using the **XIPPORTEREX & ROM** tool.

SYS Folder

The system (SYS) folder is divided into Operating System **Build Version Specific** folders. The **Build Version Specific** folders will often contain files (packages) specific to the device DPI (Dot Per Inch) and Horizontal/Vertical display size.

Each Operating System **Build Version Specific** folder is further divided into a **ROM DPI**, **ROM Resolution**, and **Shared** folder – providing a significant amount of flexibility during ROM compilation.

The **ROM DPI**, **ROM Resolution**, and **Shared** folders are further divided into **Locale Specific** folders and a **Common** folder. The **Common** folder is typically used for system files (packages) that can be installed in any locale – not specific to a language. **Locale Specific** folders are typically used for system files (packages) that should only be applied to a specific locale – for example, 0409 (English).

Tip

- The **ROM DPI** and **ROM Resolution** drop-down boxes in the Visual Kitchen permits selection of different Operating System **Build Version Specific** resolution and bit depth.

Preparing Your Facility

Before you can begin to cook your own ROM, you need to equip your facility with some Kitchen utensils. Your Kitchen is going to require a good Unicode & UTF-8 text editor; I personally use *ConTEXT* & *Notepad*. Another handy utensil to have is a comparison utility for date/file/binary comparisons; I use *WinDiff* & *BeyondCompare*. Some other utensils that you're going to require are: *Microsoft ActiveSync*, *.NET Framework 2.x/3.x*. You will also need an archive extraction utensil; I use *IZArc*, *WinRAR*, and *WinZIP*. You'll also need a good Hexadecimal calculator; I use *Windows Calculator* (Scientific Mode).

It's also a good idea to ensure that your Kitchen remains "pest" free; common pest control services include *AVG*, *McAfee*, and *Symantec* anti-Virus. You'll need to add the **RaphaelWrapper** to your list of anti-virus exclusions so that as to ensure that your anti-virus does not interfere with the ROM flashing activity.

References

CustomRUU for Raphael/RaphaelWrapper
<http://forum.xda-developers.com/showthread.php?t=410761>

To assist you in your apprenticeship, I have included a link to the Generic Visual Kitchen that I used to prepare this guide – the kitchen also includes a .DOC and .PDF format of this guide. The procedures were tested against a GSM Raphael device. I can't confirm that these procedures will work on CDMA device ROM's. Additionally, other device ROM's may not be compatible with this kitchen format.

You're going to need a **RUU_SIGNED.NBH** file; I used the following HTC Official Generic ROM – you'll need to extract the contents of the **.EXE** using an archive utensil.

[ROM] [WWE] Raphael HTC 5.05.405.1 Radio Signed (52.58.25.3 0,1.11.25.01)
http://rapidshare.com/files/193966082/RUU_Raphael_HTC_WWE_5.05.405.1_R_Radio_Signed_Raphael_52.58.25.30_1_11.25.01_Ship.rar
<http://www.megaupload.com/?d=0F50UM5K>

For the purpose of this guide, I will assume that you have added the **C:\XDA** folder, sub-folder, and files to your anti-virus exclusion list – at the very least **RaphaelWrapper.exe** – and that the contents of the Generic Visual Kitchen were extracted to the following folder.

C:\XDA\MY_VISUAL_KITCHEN

The guide is divided into the following sections:

- Extracting the RUU_SIGNED.NBH Contents
- Reducing the .PAYLOAD File
- Extracting the XIP.BIN Contents
- Increasing the Data Cache
- Unlocking the Paging Pool
- Disabling Certificate Checking
- Reducing the Update Loader (ULDR) Partition
- Changing the Unsigned CAB Policies
- Changing the Unsigned Themes Policies
- Changing the Remote API (RAPI) Policies
- Compiling the New RUU_SIGNED.NBH File
- Flashing the RUU_SIGNED.NBH File

I will attempt to provide an overview, the list of tools required, and the process to follow in each section. As you become more comfortable (and familiar) with the activities, you will find that you can consolidate (or skip) certain outlined steps. Incidentally, you'll probably want to keep these web links open in case you need to lookup some of the terms or concepts in the guide.

Acronyms

<http://wiki.xda-developers.com/index.php?pagename=Acronyms>

Glossary

<http://wiki.xda-developers.com/index.php?pagename=Glossary>

Development Resources for Windows Mobile

<http://forum.xda-developers.com/showthread.php?t=445396>

Extracting the RUU_SIGNED.NBH Contents

An **.NBH** is a signed group of modules or packages; they are typically comprised of **.NB** files. An **.NBH** can contain any combination of **.NB** files. An **.NB** file is a block of code that can be a Radio ROM, Operating System packages (XIP and IMGFS), Startup Splash Screen (or SPL).

The file we will be working with is the **OS.NB** file; it contains the ULDR, XIP, and IMGFS (OEM, SYS). To extract the contents of an **.NBH** file, we initiate the **Ervius NBH/NB/PAYLOAD Dumper** tool from within the Generic Visual Kitchen.

Upon completion, the following files will have been extracted: **OS.nb**, **OS.nb.payload**. Additionally, the **Ervius NBH/NB/PAYLOAD Dumper** tool creates a **DUMP** folder that contains all the files required.

Tools Required:

The following Ervius Visual Kitchen tool will be used for the **RUU_SIGNED.NBH** extraction activities.

Dump NBH/NB/PAYLOAD

Procedure

The following procedure initiates the ROM extraction activity via the **Ervius NBH/NB/PAYLOAD Dumper** tool built into the Generic Visual Kitchen. The extraction process can take a significant amount of time to complete.

1. Copy the **RUU_SIGNED.NBH** file to the **C:\XDA\My_Visual_Kitchen\BaseROM** folder.
2. Navigate to the **C:\XDA\My_Visual_Kitchen** folder.
3. Launch **ErviusKitchen.exe**.
4. At the multiple warning messages, click **OK**.

Warnings that may appear include:

*Folder Not Found
Could Not Find A Part Of The Path
You Need To Specify ... First*

5. Click the **Dump NBH/NB/PAYLOAD** button.
6. Navigate to the **C:\XDA\MY_VISUAL_KITCHEN\BaseROM** folder.
7. Select the **RUU_signed.nbh** file and then click **Open**.
8. At the *All Done... Nbh/nb/payload Dumped and "Kitchen" created Successful!!!* message, close the **Ervius Visual Kitchen** application.
9. Navigate to the **C:\XDA\MY_VISUAL_KITCHEN\DUMP** folder.
10. Move the sub-folders (and content) to the **C:\XDA\MY_VISUAL_KITCHEN** folder.
11. At the *Confirm Folder Replace* message, click **Yes To All**.
12. At the *Confirm File Move* message, click **Yes To All**.

- Tip**
- The **C:\XDAMY_VISUAL_KITCHEN\BaseROM\Dump** folder should be empty at this point – and can be removed.

Reducing the .PAYLOAD File

At this point, the **Ervius NBH/NB/PAYLOAD Dumper** tool has removed the contents of the IMGFS (OEM, SYS) from the **.PAYLOAD** file in preparation for changes to the ULDR and XIP. Removing the IMGFS (OEM, SYS) contents from the **.PAYLOAD** file reduces the size of the **.PAYLOAD** file making it easier to work with.

The **Ervius NBH/NB/PAYLOAD Dumper** tool has placed a copy of the reduced **.PAYLOAD** file in the **C:\XDAMy_Visual_Kitchen\ROM\Raphael** folder.

Tip

- Advanced **.PAYLOAD** file operations can be performed using the **XIPPORTEREX & ROM** tool.

References

[TUT] Manual Full XIP Porting (& MANY MORE TUTORIALS)
<http://forum.xda-developers.com/showthread.php?t=438676>

Extracting the XIP.BIN Contents

The Execute-in-place (XIP) region is an area where an application can execute code directly from ROM rather than loading it from RAM. It is possible to use the **XIP.BIN** contents from a newer version of a ROM from a different device or a newer operating system. This is typically done by chefs who are looking for the most recent versions of system files from a specific device or version of an operating system – you'll eventually do the same.

At this point, the **Ervius NBH/NB/PAYLOAD Dumper** tool has extracted the contents of the **XIP.BIN** and placed a copy of the file in the **C:\XDA\My_Visual_Kitchen\ROM\Raphael** folder.

For the purposes of this guide, we will be using the same version of the system files.

Tip

- Advanced **XIP.BIN** file operations can be performed using the **XIPPORTEREX & ROM** tool.
- You can change the ROM Date/Version using the **XIPPORTEREX & ROM** tool.

References

[TUT] Manual Full XIP Porting (& MANY MORE TUTORIALS)

<http://forum.xda-developers.com/showthread.php?t=438676>

XIP Porting Guide

<http://forum.xda-developers.com/showthread.php?t=379598>

Increasing the Data Cache

File caching improves performance and also improves power management; when an application accesses physical storage, the storage device uses much more power. The less often physical storage is accessed, the longer storage devices spend in a low-power state.

By increasing the **DataCacheSize** registry value, you effectively improve the performance of applications that are file system intensive such as database and mapping applications – which results in lower physical storage access requirements. Drastically increasing the **DataCacheSize** however, may have adverse effects and slow the device down as a result of longer auto-compaction processing.

For the purposes of this guide, we are going to increase the current **DataCacheSize** value from 4MB to 8MB.

Tools Required

The following tools are required to adjust the **DataCacheSize** value.

Unicode Text Editor
Hexadecimal Calculator

Procedure

The following procedure will change the current **DataCacheSize** value of 4MB to 8MB.

1. Launch a text editor.
2. Navigate to the **C:\XDA\MY_VISUAL_KITCHEN\ROM\Raphael\20764\OEMXipKernel** folder and open the **BOOT.RGU** file.
3. Search for the following registry key entry:

```
[HKEY_LOCAL_MACHINE\System\StorageManager\Profiles\FLASHDRV\FATFS]
```

4. Locate the following registry value underneath the key:

```
"DataCacheSize"=dword:00000800 ;2048 sectors(2048*2048=4MB)
```

5. Change the registry value to the following:

```
"DataCacheSize"=dword:00001000 ;4096 sectors(4096*2048=8MB)
```

6. Save the **BOOT.RGU** file.
7. Exit the text editor.

Tip

- Make a backup copy of the **BOOT.RGU** file before editing; delete the backup file when done.

References

Windows Mobile

<http://msdn.microsoft.com/en-us/library/aa914334.aspx>

Unlocking the Paging Pool

The Paging Pool serves as a limit on the amount of memory that can be consumed by pageable data. It includes an algorithm for choosing the order in which to remove pageable data from memory. Pool behaviour is typically determined by the OEM – Microsoft sets a default value for the paging pool, but the OEM can change that value. Applications do not have the ability to set the behaviour for their own executables or memory-mapped files.

For the purposes of this guide, we are going to apply a change to the **kitchen_build_rom.bat** to set the Paging Pool size (initially set to 6MB) to a new size of 8MB.

Tools Required

The following tools are required for the Paging Pool unlock activities.

Unicode Text Editor

Procedure

The following procedure will change the **kitchen_build_rom.bat** to set the Paging Pool size during ROM compilation.

1. Launch a text editor.
2. Navigate to the **C:\XDA\MY_VISUAL_KITCHEN\Tools** folder and open the **kitchen_build_rom.bat** file.
3. In the **Find (Search)** box, type:

```
implantxip
```

4. Add the following **implantxip** command line option:

```
-PP 8
```

5. Select the **Save** option in your text editor.
6. Select the **Exit** option in your text editor.

Tip

- Spaces are usually required between command line options; the command line option should only appear once.
- You can add output logging to file by adding the following command instructions to the end of a command line:

```
| ..\tools\mtee /+ ..\%LOG_FILE%
```

References

Change PagePool Through Hex Editing (For Diamond & Raphael)

<http://forum.xda-developers.com/showpost.php?p=2903704&postcount=5>

Paging Pool

<http://msdn.microsoft.com/en-us/library/aa915364.aspx>

Paging and the Windows CE Paging Pool

http://blogs.msdn.com/ce_base/archive/2008/01/19/Paging-and-the-Windows-CE-Paging-Pool.aspx

Hermes Page Pool Tests

<http://forum.xda-developers.com/showthread.php?t=295932>

Kernel Overview

<http://msdn.microsoft.com/en-us/library/aa909237.aspx>

Disabling Certificate Checking

During the startup process of your device, the operating system verifies that each system file against an internal certificate store to ensure that each file is signed with a trusted certificate; if the system file is not signed, the file is ignored.

To allow execution of non-signed system files, we need to disable the internal certificate store verification. Once disabled, the operating system will trust all code installed regardless of its signature. This provides more control over the code that gets installed on the device – you no longer need to load and manually sign additional certificates such as those from the **sdkcerts.cab** into the device root certificate store.

For the purposes of this guide, we are going to apply a change to the **kitchen_build_rom.bat** to disable the internal certificate store verification.

Tools Required

The following tools are required to disable the internal certificate store verification.

Unicode Text Editor

Procedure

The following procedure will change the **kitchen_build_rom.bat** to disable the internal certificate store verification during ROM compilation.

1. Launch a text editor.
2. Navigate to the **C:\XDA\MY_VISUAL_KITCHEN\Tools** folder and open the **kitchen_build_rom.bat** file.
3. In the **Find (Search)** box, type:

```
implantxip
```

4. Add the following **implantxip** command line option:

```
-NoCert
```

5. Select the **Save** option in your text editor.
6. Select the **Exit** option in your text editor.

Tip

- Spaces are usually required between command line options; the command line option should only appear once.
- You can add output logging to file by adding the following command instructions to the end of a command line:

```
| ..\tools\mtee /+ ..\%LOG_FILE%
```

References

Here's how to fully and permanently disable sign/cert checking in WM5/WM6 (+bonus)

<http://forum.xda-developers.com/showpost.php?p=2104712>

[RES] RILPHONE.DLL And "How To" With A Radio

<http://forum.xda-developers.com/showthread.php?t=481026>

Kernel Overview

<http://msdn.microsoft.com/en-us/library/aa909237.aspx>

Reducing the Update Loader (ULDR) Partition

The boot loader can accommodate multiple execute-in-place (XIP) regions where individual modules can be updated after the initial operating system image file has been written to the device – the ULDR is an example of this use. The Update Loader (ULDR) provides Flash-Over-The-Air (FOTA) capabilities permitting your carrier to issue changes such as Hotfixes over the cellular network – generally, most carriers avoid this practice.

As this is generally undesirable in a cooked ROM, since we are making modifications that a carrier Hotfix might roll back, we will reduce the partition. This will cause the device to report insufficient ULDR space to the carrier FOTA request ... and the freed up space becomes available for our uses.

For the purposes of this guide, we are going to apply a change to the **kitchen_build_rom.bat** to reduce the ULDR – effectively providing approximately 3 MB of space.

Tools Required

The following tools are required to reduce the Update Loader (ULDR) partition.

Unicode Text Editor

Procedure

The following procedure will change the **kitchen_build_rom.bat** to disable the internal certificate store verification during ROM compilation.

1. Launch a text editor.
2. Navigate to the **C:\XDA\MY_VISUAL_KITCHEN\Tools** folder and open the **kitchen_build_rom.bat** file.
3. In the **Find (Search)** box, type:

```
implantxip
```

4. Add the following **implantxip** command line option:

```
-uldr
```

5. Select the **Save** option in your text editor.
6. Select the **Exit** option in your text editor.

Tip

- Spaces are usually required between command line options; the command line option should only appear once.
- You can add output logging to file by adding the following command instructions to the end of a command line:

```
| ..\tools\mtee /+ ..\%LOG_FILE%
```

References

[TUT] Manual Full XIP Porting (& MANY MORE TUTORIALS)
<http://forum.xda-developers.com/showthread.php?t=438676>

Reduce ULDR Partition Size
<http://forum.xda-developers.com/showpost.php?p=2916649&postcount=7>

[TUT] ULDR Removal for Elf/Elfins
<http://forum.xda-developers.com/showthread.php?t=446506>

[Walkthrough] How to Port a ROM [XIP and SYS]
<http://forum.xda-developers.com/showthread.php?t=389772>

Changing the Unsigned CAB Policies

Security policies are used for configuring security settings that are then enforced with the help of security roles and certificates. They provide the flexibility to control the level of security on the device. The policies are defined globally and enforced locally in their respective components. The security policy is set during boot by executing a **.provxml** configuration file. This provisioning file is in ROM and contains the default setting specified by the OEM. The provisioning document is an Extensible Markup Language (XML) file that assigns security settings to the device.

Typically, the **mxipupdate_oemoperators_100.provxml** file contains the default security policies that will be applied to the device. To allow execution of unsigned **.CAB** files, we need to change the default security policy settings that pertain to **.CAB** files.

For the purposes of this guide, we will add or change the following unsigned policy setting ID's:

- ID 4101: Unsigned CABS Policy = 8
- ID 4102: Unsigned Applications Policy = 1
- ID 4122: Unsigned Prompt Policy = 1

*** IMPORTANT ***

Policy settings may appear in other files. You may wish to perform a search through **.TXT**, **.RGU**, **.REG**, and other **.PROVXML** files using the keywords such as: *security, policies, policy*.

Tools Required

The following tools are required to edit **.PROVXML** files.

UTF-8 Text Editor

Procedure

The following procedure will add/change the unsigned policy settings to permit the installation of unsigned **.CAB** files.

1. Navigate to the **C:\XDA\MY_VISUAL_KITCHEN\OEM\Raphael\0409\OperatorPkg** folder.
2. Verify that the **mxipupdate_oemoperators_100.provxml** is not set as *Read Only*.
3. Launch your **UTF-8 Text Editor**.
4. Use the **Search** feature to find the following section:

```
<!--337.01_SecurityPolicy-->
```

5. Ensure that the following lines are present in the section:

```
<!--337.01_SecurityPolicy-->
<characteristic type="SecurityPolicy">

...

<parm name="4101" value="8" />
<parm name="4102" value="1" />
<parm name="4122" value="1" />

...

</characteristic>
```

6. Save the changes to the **mxipupdate_oemoperators_100.provxml**; ensure that the file is saved in UTF-8 format.
7. Exit your **UTF-8 Text Editor**.
8. Verify that the **mxipupdate_oemoperators_100.provxml** is set to *Read Only*.

Tip

- Make a backup copy of the **mxipupdate_oemoperators_100.provxml** file before editing; delete the backup file when done.
- Observe the line formatting and spacing in the **mxipupdate_oemoperators_100.provxml** file.
- When saving the file, select the Save As menu option and put quotes (") around the file name so as to avoid additional file extensions.

References

Security Policy Settings for Windows 6.x Mobile-Based Devices
<http://msdn.microsoft.com/en-us/library/bb416355.aspx>

Security Policy Settings for Windows 5.x Mobile-Based Devices
<http://msdn.microsoft.com/en-us/library/ms889564.aspx>

What Do The SmartPhone Policy ID's Mean
<http://www.xs4all.nl/~itsme/projects/xda/smartphone-policies.html>

Changing the Unsigned Themes Policies

Security policies are used for configuring security settings that are then enforced with the help of security roles and certificates. They provide the flexibility to control the level of security on the device. The policies are defined globally and enforced locally in their respective components. The security policy is set during boot by executing a **.provxml** configuration file. This provisioning file is in ROM and contains the default setting specified by the OEM. The provisioning document is an Extensible Markup Language (XML) file that assigns security settings to the device.

Typically, the **mxipupdate_oemoperators_100.provxml** file contains the default security policies that will be applied to the device. If a signed theme file does not have a matching root certificate in the Software Publisher Certificate (SPC) store, the file is unsigned and is therefore not executed. To allow execution of unsigned Theme files, we need to change the default security policy settings that pertain to Theme files.

For the purposes of this guide, we will add or change the following unsigned policy setting ID's:

- ID 4103: Unsigned Themes Policy = 16

*** IMPORTANT ***

Policy settings may appear in other files. You may wish to perform a search through **.TXT**, **.RGU**, **.REG**, and other **.PROVXML** files using the keywords such as: *security, policies, policy*.

Tools Required

The following tools are required to edit **.PROVXML** files.

UTF-8 Text Editor

Procedure

The following procedure will add/change the unsigned policy settings to permit the execution of unsigned Theme files.

1. Navigate to the **C:\XDA\MY_VISUAL_KITCHEN\OEM\Raphael\0409\OperatorPkg** folder.
2. Verify that the **mxipupdate_oemoperators_100.provxml** is not set as *Read Only*.
3. Launch your **UTF-8 Text Editor**.
4. Use the **Search** feature to find the following section:

```
<!--337.01_SecurityPolicy-->
```

5. Ensure that the following lines are present in the section:

```
<!--337.01_SecurityPolicy-->
<characteristic type="SecurityPolicy">

...

<parm name="4103" value="16" />

...

</characteristic>
```

6. Save the changes to the **mxipupdate_oemoperators_100.provxml**; ensure that the file is saved in UTF-8 format.
7. Exit your **UTF-8 Text Editor**.
8. Verify that the **mxipupdate_oemoperators_100.provxml** is set to *Read Only*.

Tip

- Make a backup copy of the **mxipupdate_oemoperators_100.provxml** file before editing; delete the backup file when done.
- Observe the line formatting and spacing in the **mxipupdate_oemoperators_100.provxml** file.
- When saving the file, select the Save As menu option and put quotes (") around the file name so as to avoid additional file extensions.

References

Security Policy Settings for Windows 6.x Mobile-Based Devices
<http://msdn.microsoft.com/en-us/library/bb416355.aspx>

Security Policy Settings for Windows 5.x Mobile-Based Devices
<http://msdn.microsoft.com/en-us/library/ms889564.aspx>

What Do The SmartPhone Policy ID's Mean
<http://www.xs4all.nl/~itsme/projects/xda/smartphone-policies.html>

Customizing the User Interface of Windows Mobile Powered Devices
<http://msdn.microsoft.com/en-us/library/bb416487.aspx>

Changing the Remote API (RAPI) Policies

Security policies are used for configuring security settings that are then enforced with the help of security roles and certificates. They provide the flexibility to control the level of security on the device. The policies are defined globally and enforced locally in their respective components. The security policy is set during boot by executing a **.provxml** configuration file. This provisioning file is in ROM and contains the default setting specified by the OEM. The provisioning document is an Extensible Markup Language (XML) file that is assigns security settings to the device.

The Remote API (RAPI) enables applications that run on a desktop to perform actions on a remote Windows Embedded CE-based device. This includes the ability to manipulate the file system on the remote device, including the creation and deletion of files and directories. Additionally, the Remote API (RAPI) functions can be used to create and modify databases, either in the device's object store or in mounted database volumes. The Remote API (RAPI) applications can also query and modify registry keys as well as launch applications and invoke methods on the remote device.

Typically, the **mxipupdate_oemoperators_100.provxml** file contains the default security policies that will be applied to the device. To allow unrestricted access by remote applications to implement ActiveSync operations on Windows Mobile powered devices, we need to change the default security policy settings that pertain to the Remote API (RAPI).

For the purposes of this guide, we will add or change the following Remote API (RAPI) policy setting ID's:

- ID 4097: RAPI Policy = 1

*** IMPORTANT ***

Policy settings may appear in other files. You may wish to perform a search through **.TXT**, **.RGU**, **.REG**, and other **.PROVXML** files using the keywords such as: *security, policies, policy*.

Tools Required

The following tools are required to edit **.PROVXML** files.

UTF-8 Text Editor

Procedure

The following procedure will add/change the unsigned policy settings to permit the installation of unsigned **.CAB** files.

1. Navigate to the **C:\XDA\MY_VISUAL_KITCHEN\OEM\Raphael\0409\OperatorPkg** folder.
2. Verify that the **mxipupdate_oemoperators_100.provxml** is not set as *Read Only*.
3. Launch your **UTF-8 Text Editor**.
4. Use the **Search** feature to find the following section:

```
<!--337.01_SecurityPolicy-->
```

5. Ensure that the following lines are present in the section:

```
<!--337.01_SecurityPolicy-->
<characteristic type="SecurityPolicy">

...

<parm name="4097" value="1" />

...

</characteristic>
```

6. Save the changes to the **mxipupdate_oemoperators_100.provxml**; ensure that the file is saved in UTF-8 format.
7. Exit your **UTF-8 Text Editor**.
8. Verify that the **mxipupdate_oemoperators_100.provxml** is set to *Read Only*.

Tip

- Make a backup copy of the **mxipupdate_oemoperators_100.provxml** file before editing; delete the backup file when done.
- Observe the line formatting and spacing in the **mxipupdate_oemoperators_100.provxml** file.
- When saving the file, select the Save As menu option and put quotes (") around the file name so as to avoid additional file extensions.

References

Security Policy Settings for Windows 6.x Mobile-Based Devices
<http://msdn.microsoft.com/en-us/library/bb416355.aspx>

Security Policy Settings for Windows 5.x Mobile-Based Devices
<http://msdn.microsoft.com/en-us/library/ms889564.aspx>

What Do The SmartPhone Policy ID's Mean
<http://www.xs4all.nl/~itsme/projects/xda/smartphone-policies.html>

Remote API (RAPI)
<http://msdn.microsoft.com/en-us/library/aa920177.aspx>

Compiling the New RUU_SIGNED.NBH File

Brief Review

At this point, we have increased the Data Cache size, ensured that the **implantxip** command unlocks the Paging Pool and sets an initial size 8MB, disables Certificate checking, and reduces the ULDR. We changed the Unsigned **.CAB** policy, changed the Theme policy, and changed the Remote API (RAPI) policy.

This guide did not go into details about adding, changing, or removing ROM packages to/from the OEM and SYS folders. Additionally, the guide did not cover the steps required to port an XIP from a different ROM version or device – you'll eventually want to learn how to do these types of activities.

Compiling the ROM

We now need to assemble all of these changes into a new **RUU_SIGNED.NBH** file – this process is often referred to as ROM compilation. We are going to initiate the **Create ROM** tool from within the Generic Visual Kitchen to perform this process – there are newer versions of the **Ervius Visual Kitchen** available.

Tools Required

The following Ervius Visual Kitchen feature will be used for the **RUU_SIGNED.NBH** file creation activities.

Create ROM

Procedure

The following procedure initiates the ROM creation activity via a script that is included in the Generic Visual Kitchen. The creation process can take a significant amount of time to complete and requires user interaction at various stages.

1. Navigate to the **C:\XDA\MY_VISUAL_KITCHEN** folder.
2. Launch **erviuskitchen.exe**.

3. Click on the **Device** drop-down box and select **Raphael** from the list.

In the **ROM Ver** box, change the value: 5.05.405.1

Click the **Create ROM** button, the command window will begin to display processing information after a few minutes.

Wait for the *All Done* message to appear in the **Bottom Status** window.

4. Exit the **Ervius Visual Kitchen** application.
5. Copy the **RaphaelWrapper.exe** file from the **C:\XDAMY_VISUAL_KITCHEN\Tools** folder to the **C:\XDAMY_VISUAL_KITCHEN\RELEASE_Raphael** folder.

Tip

- Processing information may take a few minutes before it appears in the Command window; this is a result of logging activity being spooled to logging files for troubleshooting purposes.
- If the compilation is successful, the last three (4) lines in the **Status** window of the **Ervius Visual Kitchen** will display:
ROM Builded successfull!!! Reenabling all skipped packages. All packages Reenabled. All done!!!

References

Visual multilang/multidevice/multibuild kitchen for last bepe rom-tools!!!
<http://forum.xda-developers.com/showthread.php?t=469420>

Flashing the RUU_SIGNED.NBH File

If you are reading this, you must have successfully compiled your **RUU_SIGNED.NBH** file and are now ready to flash it to your device ... congratulations! Now for the disclaimer bit; I take no responsibility and will not be held liable for any problems you encounter with your device before and after following this guide ... flashing a ROM is done at your own risk.

Also keep in mind that you'll need to unlock your device before venturing into the world of ROM flashing. If you haven't unlocked your device and applied the Hard SPL, you need to complete these tasks before you can proceed.

The ROM Flashing process is relatively straight forward. You need to ensure that you can connect to your device via **ActiveSync** the application. Once you have connected to the device, launch the **RaphaelWrapper** tool to flash your cooked ROM to your device – I suggest temporarily disabling your anti-virus software to avoid unforeseen problems.

Tools Required

The following tools are required to flash the **RUU_SIGNED.NBH** file to your device.

RaphaelWrapper

Procedure

The following procedure initiates the ROM Flashing activity. The flashing process can take a significant amount of time to complete and requires user interaction at various stages.

1. Navigate to the **C:\XDA\MY_VISUAL_KITCHEN\Tools** folder.
2. Copy the **RaphaelWrapper.exe** application to the **C:\XDA\MY_VISUAL_KITCHEN\RELEASE_Raphael** folder.
3. Navigate to the **C:\XDA\MY_VISUAL_KITCHEN\RELEASE_Raphael** folder.
4. Launch **RaphaelWrapper.exe**.
5. At the **Welcome To The ROM Update Utility** window, select the *I Understand The Caution Indicated* option.
6. Click the **Next** button.
7. At the **Follow The Instructions Below** window, select the *I Completed The Steps Indicated Above* option.
8. Click the **Next** button.
9. At the **Current Information About Your PDA Phone** window, click the **Update** button.
10. At the **Verify That You Want To Update The ROM Version** window, click the **Next** button.
11. At the **You Are Now Ready To Update Your ROM Image** window, click the **Next** button.
12. Click the **Finish** button.

Tip

- The current version of the **PDA Phone ROM Update Utility 3.27.4.3** we are using has been modified to not format the device when only flashing a Radio, Startup Screen, or Operating System.
- You can usually recover from a failed flash by copying a known "working" **.NBH** file to a MicroSD Card and starting your device in the Tricolour bootloader.

References

CustomRUU for Raphael

<http://forum.xda-developers.com/showthread.php?t=410761>

[Resources] Flashing your First GSM Raphael Rom (For Noobs)

<http://forum.xda-developers.com/showthread.php?t=448008>