

# Forensic Image Analysis of Familiar-based iPAQ

Cheong Kai Wee

School of Computer and Information Science, Edith Cowan University  
ckw214@yahoo.com

Lih Wern Wong

School of Computer and Information Science, Edith Cowan University  
lihwern@yahoo.com

## Abstract

*One of the PDA forensic issues is the difficulty in maintaining image integrity as two consecutive acquisitions on most PDA platforms could produce different images. JFFS2 used in certain Linux PDA further complicate the forensic process as compression is implemented in this file system. Currently there is not specific tool that can analyse this file system. This paper analyse the integrity of the acquired Familiar iPAQ images. Experiments are also carried out to analyse the possibility of file recovery on JFFS2.*

## Keywords

Linux PDA, Familiar, JFFS2, Forensics

## INTRODUCTION

PDA's are commonplace in today's society. Initially these devices were used as PIM, equipped with common appointments application, to-do list, calendars and address books. However, these devices are continuously evolving as new technology emerges and other technologies converge with PDA's. An example of this would be the marriage of PDA and mobile phone. The computational and processing power of PDA has improved significantly (Valli, 2005). With the extension slot on PDA that supports removable storage technology such as Secure Digital and Compact Flash cards, PDA can easily store several gigabytes of data.

PDA's are commonly used in business organisation, both with and without permission. In either case, PDA presents the risk of confidential data being copied and transfer out of the organisation. Due to the size and ease of concealment of these devices, it would be hard to detect unauthorised use of PDA. Security breaches mitigation and recovery through forensic investigation is the possible last best hope to protect the organisation (Valli, 2005). However, performing forensic acquisition on PDA without modifying its integrity is not easy, if not impossible.

This paper focuses its discussion on Linux PDA, specifically Familiar-based iPAQ. The aim of this paper is to perform investigation on how certain actions such as soft reset and hard reset could affect the integrity of the Familiar's images. The acquired images are also analysed for possible deleted file recovery on Familiar-based PDA.

## PDA BACKGROUND

PDA's are small and mobile handheld devices for organising personal data such as telephone numbers, appointments and notes. They usually contain a microprocessor, ROM (Read Only Memory), RAM (Random Access Memory), a variety of hardware keys and interfaces, and a LCD (Liquid Crystal Display) screen. PDA's are powered by battery and usually support different types of expansion sleeves and networking interfaces. Figure 1 shows a generic hardware diagram of handheld devices.

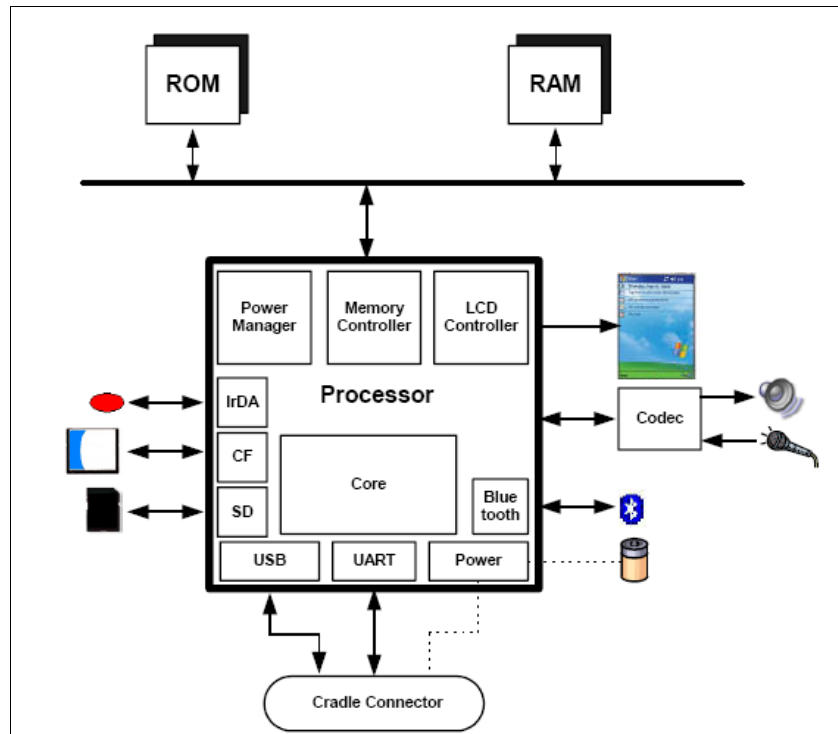


Figure 1: Generic Hardware Diagram of Handheld Devices.

### PDA Generic State

As shown in Figure 2, there are 4 generic states on PDA as opposed to desktop computers which are either in “on” or “off” state (Jansen & Ayers, 2004). These 4 states are:

- Nascent state: This is the state when the PDA is first released from the manufacturer. It is configured with default factory setting with no user data. This state can be attained again by hard resetting the device or letting the PDA to completely drain off its power.
- Active state: PDA in active state is powered on and performing task. After a soft reset and working memory re-initialization, PDA returns to active state.
- Quiescent state: This is a power reserved mode that conserves battery life while maintaining user data and performing other background functions. Context information is preserved in memory to allow quick resumption of processing.
- Semi-active mode: This is a state between active and quiescent state. It is usually triggered by timer and it preserves the battery life by dimming the display and taking other appropriate actions. Some PDAs may not support this state.

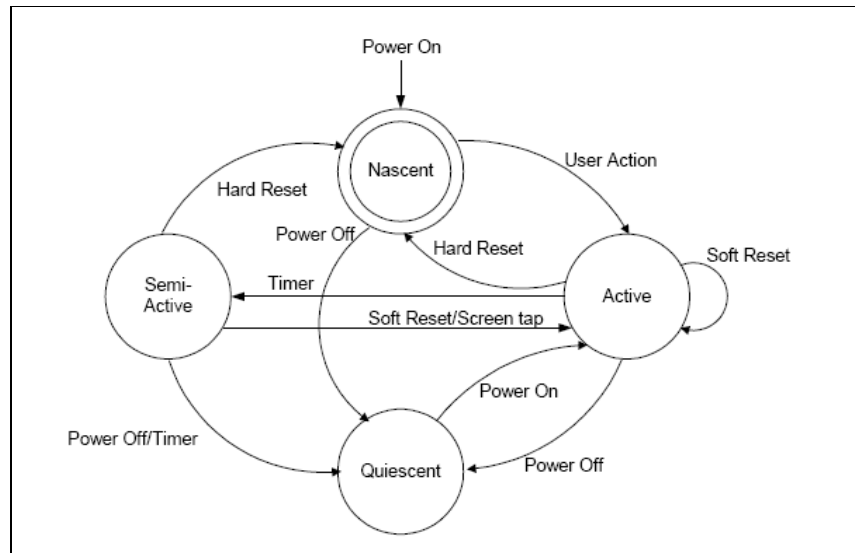


Figure 2: PDA Generic States

### PDA and Forensics Process

Forensic analysis on handheld devices presents the following issues:

- Forensic investigator need to maintain the power of PDA to obtain the content in the memory.
- PDA is not standardised. There are variety of platforms and operating systems, which include Window, Linux, Palm, Java and others (Informat, 2005).
- The integrity of acquired images is hard to maintain. For most PDA, it is highly unlikely for two consecutive acquisitions to be forensically identical (Frichot, 2004).
- The tendency of frequent garbage collection execution and memory reorganization on PDA devices to make space for new files causes difficulty in file recovery (Valli, 2005).

This paper discusses the third and fourth issues. Experiments is carried out on a iPAQ handheld device with Familiar distribution to analyse how certain actions such as soft reset and hard reset could affect the integrity of image. The images is also analyse for possible file recovery after garbage collection is started.

### POCKET PC

Pocket PC is Microsoft version of PDA device which runs on Windows CE operating system with additional functionality. Pocket PC is also known as Windows Mobile operating system. Pocket PC runs on various type of processors, including ARM, Intel Xscale, MIPS and SH3 processors. All Pocket PC devices have ROM and RAM. The RAM is used to store user files, PIM entries and additional installed applications, while ROM is used to store operating system image and supporting applications that are bundled with Pocket PC. In newer version of Pocket PC, unused space in ROM can be used to backup files stored in RAM. Most Pocket PC supports card slots such as Compact Flash (CF) and Secure Digital (SD) to provide additional functional capabilities.

Files can be stored in RAM and ROM. In fact, Window CE supports objects or files stored in RAM, ROM, and installed file systems. When a file stored in RAM has the same name as the one stored in ROM, file stored in RAM supersedes the ROM version (Intel, 2005, p. 81). User will only have access to the RAM version until the RAM copy is deleted. This feature is usually used in upgrading existing ROM-based application files.

Windows CE operating system supports the following four type of memory.

- RAM
- Expansion RAM
- ROM
- Persistent Storage

However, only RAM and ROM are required for building a minimal functional Pocket PC.

## **RAM**

Pocket PC main memory, RAM is organized into two separate areas, Object Store and Program Memory. Depending on Pocket PC manufacturer, end user may be allowed to modify the Object Store and Program Memory allocation via application level control without rebooting.

Program Memory can be analogised as RAM in normal desktop PC. It is used to store running processes. Program Memory is managed and allocated in a paged virtual memory system form. Windows CE will handle the mapping of virtual memory addresses and physical memory locations. Applications and files stored in Object Store are compressed. Upon execution, the application is paged down and decompressed in Program Memory for execution, a process known as demand paging.

Object Store which occupies the other portion of the RAM is classified into three categories, namely File System (refers to files and directories stored in RAM), Registry and Database Objects. Registry is similar to desktop PC Windows registry, which is used to store applications configuration, users preference and drivers. The Registry is stored and run in RAM. If the Registry is lost or corrupted, the original default ROM version is loaded. Database Objects refers to entries such as contact list, phone numbers, or entries in PIM applications. Database files can be stored compressed and uncompressed. Windows CE also allows the mounting and managing of Database Objects in installed file systems (e.g. FAT) on external media (Intel, 2005, p. 82).

### **Expansion RAM**

Expansion RAM allows additional RAM to be added to provide extra storage and memory. After a cool reboot, the extended RAM is mapped to the virtual memory and appears functional identical in the virtual memory map as system RAM.

## **ROM**

ROM contains boot loader, Windows CE image file and default applications files. Files in ROM are usually compressed and decompressed in RAM upon execution. Since Windows CE supports XIP (eXecution In Place), some executables and DLLs are left uncompressed and page-aligned to enable XIP (Intel, 2005). During Windows CE image build process, individual files operation form can either be XIP or paged on demand (for compressed files). Windows CE only supports a complete OS image update on the flash ROM, but not modular ROM image update. Subsequent patches on applications files are stored in RAM.

### **Persistent Storage**

Windows CE uses FAT file systems and Installable file systems (IFS) to support persistent storage (non-volatile storage), which allows users to backup and restore files (e.g. executables, user files, registry, database objects) to and from persistent storage. Persistent storage usually refers to external storage cards (i.e. Compact Flash, Secure Digital). Files stored in persistent storage are stored uncompressed and mapped into the system RAM. With the aid of 3<sup>rd</sup> party product, Intel Persistent Storage Manager allows user to use the remaining ROM memory for persistent storage (Intel, 2005).

## **PALM OS**

The Palm OS and built-in application are stored in ROM while other applications and user data are stored in RAM. Users can use certain add-on utilities to backup the PIM data to the ROM (Jansen & Ayers, 2004).

The memory architecture of Palm OS divide the entire RAM into two logical areas: dynamic RAM and storage RAM as shown in Figure 3. The dynamic RAM is used as a working space for application, which is similar to RAM in desktop computers to store data such as global variables, system dynamic allocations, application stacks and temporary memory allocations. The rest of the RAM is allocated to storage RAM to store user data such as appointments, to do lists, memos and address lists. The size of the storage heap varies according to OS version, amount of RAM and ROM available, and also storage requirement of end users application (Wilson, 2004, p. 4). As long as power is constantly applied to the RAM, both areas of the RAM preserve their contents. If the Palm is soft reset, dynamic RAM is reinitialised while storage RAM remains intact. Hard reset reinitialises both areas and user data in storage RAM is completely erased (PalmSource, n.d.).

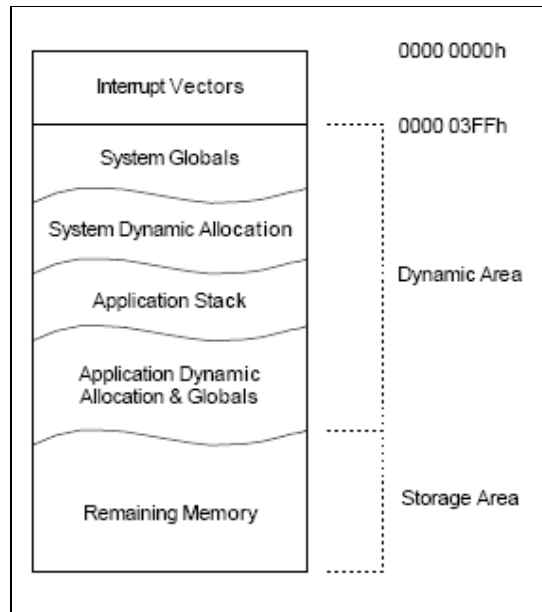


Figure 3: RAM Usage of Palm OS (Grand, 2002)

The Palm File Format Specification defines 3 types of files (Hillerman, 2003):

- a) Palm Database (PDB): Record databases used to store user data or application.
- b) Palm Resources (PRC): a database similar to Palm database storing application code and user interfaces object.
- c) Palm Query Application (PQA): Palm database containing WWW content.

These databases are similar to files. The only different is that they are stored in memory chunks called records or resources instead of one continuous area (Grand, 2002).

The core memory architecture in Palm OS is the concept of card (Mislán, n.d.). Card is a logical entity and is not necessarily corresponding to a physical device (PalmSource, n.d.). Palm OS is designed to be modular in that multiple cards can be used in a single device and each card can contain only RAM, ROM or both (Grand, 2002).

## LINUX PDA

Besides Windows CE and Palm OS, Linux can also be installed as an operating system for handheld devices. The success of Linux PDA rests on its open source model and wide support of useful and free applications.

Figure 4 shows the conceptual architecture of Linux operating system. Linux operating system is responsible for memory management, process and thread creations, inter-process communication, interrupt handling, file system management and networking interface (Bowman, Siddiqi & Tanuan, 1998).

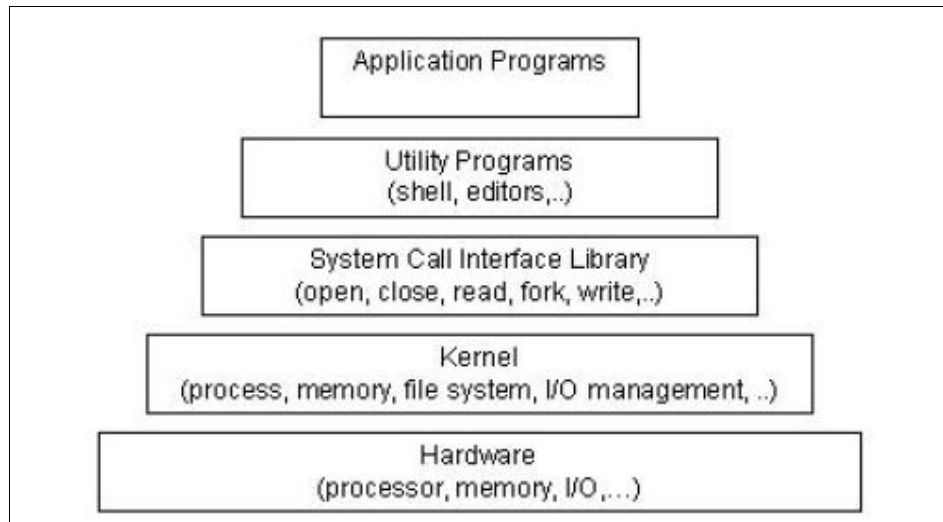


Figure 4: Linux Architecture (Jansen & Ayers, 2004).

The boot loader is the process that initialises hardware, memory and loads the kernel. Linux based kernel usually provides a rich command set and accept kernel image transfer over different interfaces such as serial connection, Ethernet connection and memory cards (Jansen & Ayers, 2004).

There are many Linux distributions for PDA. This includes Zaurus, Familiar and Embedix Linux. Different distributions use different memory management approach and different file system. For example, Familiar use JFFS2 and Zaurus use ext2. The Linux distribution used in this experiment is Familiar.

### **Familiar Distribution**

Familiar is a popular Linux distribution for iPAQ. Familiar is different from other PDA operating system which stores data in RAM, all data in Familiar is stored in flash ROM or removable memory card. These include documents, settings, PIM data and installed software (PaulEggleton, 2005). Due to this feature, data is not loss when the PDA battery is depleted or the PDA is soft or hard reset.

There are two types of user environment for Familiar, which are Opie and GPE. Both of them provide fully graphical user environment and wide range of application suite. Opie is chosen as the user environment for the iPAQ.

Default installation of Familiar includes support for SSH, JFFS2, Python and XFree86. Familiar uses JFFS2 as file system to store files in ROM, as opposed to ext2 used in Zaurus and other Linux PDA. Familiar also includes a packaging system called ipkg which manage installation, update and removal of packages.

### **JOURNALING FLASH FILE SYSTEM, VERSION 2 (JFFS2)**

Familiar stores user files and applications in ROM instead of the storage area in RAM, as featured in Windows Pocket PC and Palm OS. Hence, a hard reset (cold reboot) on Familiar will not erase user files and external applications.

Familiar uses JFFS2 (Journaling Flash File System, version 2), a log-structure file system which is the successor to JFFS (version 1). JFFS2 is used in flash memory devices, such as PDA's ROM. JFFS was designed to manage flash memory usage effectively, by taking into account the characteristics of flash memory, and the high possibility of improper shutdown in embedded system devices (Yaghmour, 2003). Flash memory chips are arranged in block. When resetting bits from zero to one, also known as "erasing", it will erase the entire block instead of the individual bits (Woodhouse, 2001).

Furthermore, flash chips have a lifetime which is measured in the number of erase cycles. Each block has a lifespan of approximately 100,000 erases. A process known as "wear leveling" is used in JFFS to ensure all areas of the memory are evenly used. JFFS2 for instance will prevent the erase of the entire block to rewrite a single byte. Standard file systems such as ext2 does not provide wear leveling and power-down-safe file system which suits the flash medium characteristics (Malik, 2001). In standard block based file system, files changes are stored in block.

Before JFFS was introduced, most embedded system with flash media employs a translation layer such as FTL or NFTL to emulate block devices with 512-byte sector, a common sector size in standard file systems. Although the translation layer is a form of journaling file system and it provides wear leveling, the use of a standard file system on top of the extra translation layer is inefficient. Hence, JFFS was introduced to efficiently use the flash media without the need of the translation layer (Woodhouse, 2001).

JFFS2 records changes to files and directories using nodes. Each JFFS2 raw nodes have inode number which indicates which inode the nodes belong to. Each node contains file system metadata for that particular inode, and if applicable the associated actual data. These nodes that belong to the same inode have version numbers to indicate the newest node for that inode (Axis, 2001).

There are two types of nodes in JFFS2, *inode* and *dirent node*. *Inode* contains files' metadata, and the actual data if applicable. The data that is associated to the inode is compressed using for instance the zlib compression library (Woodhouse, 2001). *Dirent node* refers to directory entries, which contains link to inode. Each *dirent node* contains parent (directory) inode number, name of the link and the inode number of the inode the link refers to. A version number in dirent node indicate the sequence of the parent inode. When a link is removed from the directory entries, JFFS2 create a new dirent node with same parent inode number and link name, but with target inode number equals to zero and a different version number. When nodes are initially created, they are known as *valid nodes*. However, they are *obsolete* when newer version of the node is created. Space taken by obsolete nodes is referred as *dirty* space.

During system start up, all nodes in the medium are scanned at file system mount time. All nodes are read and interpreted to reconstruct the file system structure, and the map for each inode to its respective physical memory location (Woodhouse, 2001).

Unlike JFFS which treat the flash medium as a single circular log, JFFS2 deals in blocks, which is the same size as erase segment of the flash medium. Each erase block can be in various states depending on the nodes inside the block. JFFS2 keeps a list of the blocks in various states. Erase blocks in *clean\_list* contain only valid nodes. Dirty blocks in *dirty\_list* contain at least one obsolete node. Free blocks in *free\_list* contain only one valid node, which indicate that the block has been properly and completely erased.

JFFS2 always starts writing node from the beginning of an erase block and sequentially until the block is filled. Subsequent writes are performed on new blocks that are selected from the *free\_list*. When the size of *free\_list* reaches certain threshold, garbage collection process starts (Woodhouse, 2001). Using some probabilistic method, garbage collection is usually performed on blocks in *dirty\_list* (Woodhouse, 2001). Garbage collector will convert dirty blocks into free blocks. Valid nodes in dirty blocks will be copied to new blocks until the whole block can be erased without losing the valid nodes (Brake, 2001). Occasionally, 1 in 100 times, garbage collector will consume free blocks to ensure wear-leveling, especially on mostly-static files in the system. Garbage collections however require some spare space on the medium to function properly (Yaghmour, 2003). Currently, there is no formal way to determine the minimum required space for garbage collection. A conservative approach is suggested, where five free erase blocks should be made available for new writes in garbage collection (Woodhouse, 2001).

Since compression is used to stored data, JFFS2 could not allow XIP (eXecution In Place). XIP allows the execution of codes directly from ROM without the need to copy them to RAM (Yaghmour, 2003). JFFS2 will decompress the files to RAM before executing them, and recompress them before storing it back to ROM.

The fact that compression is used in reducing the size of the file system data (not the metadata), and without the aid of automated tools to interpret and decompress the file system, manual forensic analysis on the JFFS2 images is tedious and difficult.

## FORENSIC TOOLS FOR LINUX PDA ACQUISITION

Unlike the situation on desktop computer, acquisition tools for handheld devices are limited. Common tools used for forensic acquisition of PDA include pdd, Pilot-Link, PDA Seizure, EnCase and dd. As shown in Table 1, the only forensic acquisition tools for Linux PDA is dd.

	<b>Palm OS</b>	<b>Pocket PC</b>	<b>Linux PDA</b>
Pdd	Acquisition	NA	NA
Pilot-link	Acquisition	NA	NA
PDA Seizure	Acquisition, Examination, Reporting	Acquisition, Examination, Reporting	NA
EnCase	Acquisition, Examination, Reporting	NA	Examination, Reporting

dd	NA	NA	Acquisition
----	----	----	-------------

Table 1: PDA Forensic Acquisition Tools (Ayers & Jensen, 2004)

### dd (Duplicate Disk)

dd utility creates a bit perfect image of a device. Unlike other tools such as pdd, PDA Seizure and EnCase, dd executes directly at the PDA and must be invoked through a remote connection or command line input (Jansen & Ayers, 2004). The image acquired can be dumped into an auxiliary device such as memory card or transferred across the network to a forensic workstation. dd creates raw image as opposed to proprietary embedded image created by EnCase (Carrier, 2005). Raw image is flexible and can be imported to different analysis tools easily.

The experiments in this paper are performed using dd. Images are transferred to forensic workstation using netcat utility. /dev/mtd/1, iPAQ flash ROM which stores the Linux file system is acquired instead of /dev/root as it generates input/output error.

## HARDWARE AND SOFTWARE USED IN EXPERIMENT

All experiments in this paper are performed on a iPAQ 3970, with the following OS details:

- a) Familiar Linux v0.8.2
- b) Kernel version 2.4.19-rmk6-pxal-hh37
- c) Opie v1.2.0 as user environment in Familiar

Specification of iPAQ 3970:

- a) 64Mb RAM / 48MB ROM
- b) 400 MHz processor
- c) 3.8" TFT colour screen 320x240 pixels
- d) Handwriting recognition/Stylus/On-screen soft keyboard
- e) Serial or USB sync cable
- f) Infrared port IR and IrDA (115 Kbps)
- g) Bluetooth
- h) Microphone, Speakers, Voice Recorder
- i) Dimensions: 5.3" x 3.3" x .6"
- j) Weight 6.5oz / 184g
- k) Average battery life 12-24 hrs, 1400 mAH Lithium Polymer
- l) SD memory card slot
- m) Compact Flash type I/II, PCMCIA card, GPS, GPRS, VGA, modem, wireless and wired networking, etc. as expansion jacket add-ons

Desktop operating system and tools used in image acquisition and analysis:

- a) Knoppix-STD 0.1 (kernel version 2.4.21). The kernel is compiled to support JFFS2 file system, loopback device and mtdblock to enable image analysis in one of the following experiments.
- b) netcat v1.10-27
- c) dd v5.0
- d) md5deep v0.16
- e) Foremost 0.69
- f) Hexedit 1.2.2-1
- g) WinHex 12.6 (installed in Windows XP)

## IMAGE INTEGRITY TESTING

Four experiments are carried out to test the image integrity, which are:

- a) Image integrity test for /dev/mtd/1
- b) Image integrity test for /dev/mtd/3
- c) Test to uncover possible file change after soft reset
- d) Test to uncover possible file change after hard reset

### Image Integrity Test for /dev/mtd/1

/dev/mtd/1 refers to iPAQ 32MB ROM storage. The purpose of this test is to verify the integrity of the acquired image on /dev/mtd/1. The image is acquired at various stages, including before soft reset, after soft

reset and after hard reset. The package `mtd-utils_20050217-r2_arm.ipk` needs to be installed into Familiar to unlock `/dev/mtd/1` via the following command:

```
ipkg install mtd-utils_20050217-r2_arm.ipk
```

Without the above package, acquisition on Familiar ROM will generate an input/output error.

Step 1: Acquire initial image

- 1) Connect iPAQ to the desktop computer with serial connection.
- 2) Boot Familiar operating system on the iPAQ.
- 3) Initiate a PPP connection from desktop computer to the iPAQ with the following command:
 

```
/usr/sbin/pppd /dev/ttyS1 115200 192.168.1.100:192.168.1.3
nodetach local noauth nocrtscts lock user ppp connect
"/usr/sbin/chat -v -t3 ogin--ogin: ppp"
```
- 4) Initiate a SSH connection to the iPAQ with the following command:
 

```
ssh 192.168.1.3
```
- 5) Load mtd interface and unlock the ROM to be acquired:
 

```
modprobe mtdchar
flash_unlock /dev/mtd/1
```
- 6) Configure netcat at the desktop computer to listen to port 9000:
 

```
nc -v -l -p 9000 | dd of=mtd1_default.dd
```
- 7) Calculate MD5 checksum of the original image
- 8) Acquire the image using dd:
 

```
dd if=/dev/mtd/1 | nc 192.168.1.100 9000
```
- 9) Calculate MD5 checksum of the created image

Step 2: Acquire second image without resetting

Wait for 30 minutes and repeat the step 6-9 above to acquire the second image. Save output image as `mtd1_withoutreset.dd`. The waiting interval is used to confirm the image integrity after a period of time.

Step 3: Acquire third image after resetting

Soft reset the iPAQ. Repeat the above steps and save the output image as `mtd1_aftersoftreset.dd`.

Step 4: Acquire fourth image after hard reset

Hard reset the iPAQ. Repeat the above steps and save the output image as `mtd1_afterhardreset.dd`.

## Findings

Although Familiar stores all files in ROM, the integrity of image cannot be maintained after soft reset as shown in Table 2. However, the image integrity remains intact when the device reacquire after 30 minutes. This shows that it is possible to maintain image integrity if no soft or hard reset is performed. Maintaining image integrity would not be possible if the acquisition is done on a Palm OS device, even though the acquisition is done consecutively without any form of rebooting (Frichot, 2004). One of the reasons that changes Palm OS and Windows CE image integrity on consecutive acquisition is OS clock issue, which may be applicable to Familiar. However, Familiar stores current clock data in RAM. A soft reset on Familiar reverts the OS date-time to the default value confirms date-time retention in RAM.

Images	MD5 Checksum of Acquired Image
<code>mtd1_default.dd</code> (default)	35ff901662e458a59ea9ebf783a1781a
<code>mtd1_withoutreset.dd</code> (wait 30 minutes, without reset)	35ff901662e458a59ea9ebf783a1781a
<code>mtd1_softreset.dd</code> (soft reset)	4d5c6126d6f00abf22a09de4a29fa01f
<code>mtd1_hardreset.dd</code> (hard reset)	a5749d9227492119585b85512de45705

Table 2: MD5 Checksum of Acquired Images

## Image Integrity Test for `/dev/mtd/3`

iPAQ 3970 has 48 MB of flash ROM, but only 32 MB are used by the Familiar operating system in `/dev/mtd/1`. The remaining 16 MB is unused by Familiar and can be located at `/dev/mtd/3`. User has to manually mount it to use the remaining 16 MB. This experiment test the integrity of `/dev/mtd/3` after soft reset and hard reset.

Step 1: Acquire initial image

- 1) Connect iPAQ to the desktop computer with serial connection.

- 2) Boot Familiar operating system on the iPAQ.
- 3) Initiate a PPP connection from desktop computer to the iPAQ with the following command:
 

```

/usr/sbin/pppd /dev/ttyS1 115200 192.168.1.100:192.168.1.3
nodetach local noauth nocrtscts lock user ppp connect
"/usr/sbin/chat -v -t3 ogin--ogin: ppp"

```
- 4) Initiate a SSH connection to the iPAQ with the following command:
 

```
ssh 192.168.1.3
```
- 5) Load mtd interface and unlock the ROM to be acquired:
 

```

modprobe mtdchar
flash_unlock /dev/mtd/3

```
- 6) Configure netcat at the desktop computer to listen to port 9000:
 

```
nc -v -l -p 9000 | dd of=mtd3_default.dd
```
- 7) Calculate MD5 checksum of the original image
- 8) Acquire the image using dd:
 

```
dd if=/dev/mtd/3 | nc 192.168.1.100 9000
```
- 9) Calculate MD5 checksum of the acquired image

Step 2: Acquire second image without resetting

Wait for 30 minutes and repeat the step 6-9 above to acquire the second image. Save output image as `mtd3_withtoutreset.dd`.

Step 3: Acquire third image after resetting

Soft reset the iPAQ. Repeat the steps above and save the output image as `mtd3_softreset.dd`.

Step 4: Acquire fourth image after hard reset

Hard reset the iPAQ. Repeat the steps above and save the output image as `mtd3_hardreset.dd`.

## Findings

The checksum of `/dev/mtd/3` remains the same throughout the experiments as shown in Table 3. Since, the 16 MB ROM based storage is not used by Familiar, MD5 checksums on all the images should remain the same.

Images	MD5 Checksum Of Acquired Image
<code>mtd3_default.dd</code> (default)	4d5b077432f48ddf8d76001e029b15af
<code>mtd3_withoutreset.dd</code> (wait 30 minutes, without reset)	4d5b077432f48ddf8d76001e029b15af
<code>mtd3_softreset.dd</code> (soft reset)	4d5b077432f48ddf8d76001e029b15af
<code>mtd3_hardreset.dd</code> (hard reset)	4d5b077432f48ddf8d76001e029b15af

Table 3: MD5 Checksum of Acquired Images

## Test to Uncover Possible File Change After Soft Reset

Since Familiar stores files in ROM, it is reasonable to believe that a soft reset should not modify the ROM content. The purpose of this test is to uncover files that are changed after soft reset. MD5 value is calculated using `md5deep` on all the files in the image to uncover possible file change. To enable such image analysis, the Linux analysis machine must be compiled to support JFFS2 and MTD (Memory Technology Devices). Refer *Familiar Backup Howto* on the required modules during kernel compilation and commands to mount JFFS2 images (TobyGray, n.d.). The procedures for this test are as following:

- 1) Mount `mtd1_withoutreset.dd` to `/mnt/ipaq3900` and calculate MD5 hashes for all files in `mtd1_withoutreset.dd` using `md5deep`:
 

```
md5deep -re /mnt/ipaq3900 > md5_mtd1_withoutreset.txt
```
- 2) Mount `mtd1_aftersoftreset.dd` and calculate MD5 hashes for all files in `mtd1_aftersoftreset.dd` using `md5deep`:
 

```
md5deep -re /mnt/ipaq3900 > md5_mtd1_aftersoftreset.txt
```
- 3) Compare MD5 hashes for files in `mtd1_withoutreset.dd` and `mtd1_aftersoftreset.dd` using `diff`

```
diff md5_mtd1_withoutreset.txt md5_mtd1_aftersoftreset.txt
```

## Findings

The only file change is `/var/run/utmp` in iPAQ. This file contains login records (e.g. login time), which changes each time login is required to acquire iPAQ image. Content of this file changes when SSH connection is used to connect to iPAQ. In other words, if acquisition is performed by accessing the flash ROM directly with

suitable ROM reader, the integrity of image should be maintained. However, due to lack of appropriate equipments, further experiment could not be done to confirm this statement.

### Test to Uncover Possible File Change After Hard Reset

The purpose of this test is to uncover files that are changed after hard reset. The procedures for this test are as following:

- 1) Mount `mtd1_withoutreset.dd` to `/mnt/ipaq3900` and calculate MD5 hashes for all files in `mtd1_withoutreset.dd` using `md5deep`:  
`md5deep -re /mnt/ipaq3900 > md5_mtd1_withoutreset.txt`
- 2) Mount `mtd1_afterhardreset.dd` and calculate MD5 hashes for all files in `mtd1_aftersoftreset.dd` using `md5deep`:  
`md5deep -re /mnt/ipaq3900 > md5_mtd1_afterhardreset.txt`
- 3) Compare MD5 hashes for files in `mtd1_withoutreset.dd` and `mtd1_aftersoftreset.dd`, using  
`diff md5_mtd1_withoutreset.txt md5_mtd1_afterhardreset.txt`

### Findings

The result obtained is the same as the above image after soft reset, only file `/var/run/utmp` is changed.

## IMAGE ANALYSIS AND DATA RECOVERY

This experiment analyse Familiar images for possible file recovery after deleting the actual file.

Step 1: Load the test files into Familiar

The following 6 files, one PIM Contact, and one PIM Todo List entries are copied and created in Familiar. The files are transferred into Familiar using command, `scp <filename> root@192.168.1.3`

Filename	File Size (kilo bytes)
test.htm	28KB
test.doc	39KB
test.gif	30KB
test.pdf	232KB
test.jpg	140KB
test.mp3	212KB

PIM Contact

Field	Value
Full Name	John Smith
Home Phone	61898765432
Job Title	System Admin
Organization	Private Ltd.

PIM Todo List

Field	Value
Summary	Business Meeting at 10am

Step 2: Acquire image before deleting files and PIM entries.

After loading the files and PIM entries into Familiar, `/dev/mtd/1` is acquired, and output is saved as `recovery_beforedelate.dd`.

Currently, there is no automated tool to interpret and analyse the JFFS2 file systems. Since compression is used, ordinary file recovery using file header and footer signature may not be effective. However, data craving technique using Foremost is still used to detect possible file signature that are uncompressed. Foremost is configured to detect file signature for all the six file types, including mp3 (using file header `\x49\x44\x33`). `x494433` decimal value is "ID3", which is a common signature in mp3 files.

After examining `recovery_beforedelate.dd` image, Foremost could only detect the present of gif file and mp3 file. Foremost could only extract portion of the gif file. Visual inspection confirms the gif file present.

The portion of mp3 files extracted by Foremost is not playable on Winamp. However, further examination on a fresh Familiar installation (without any mp3 file), indicates the present of signature x494433. The extract mp3 file is a false positive. Hence, the extracted snippet is not used as mp3 file signature in this experiment.

For the purpose of generically identifying the six files present in the file systems after loading them, the following file signatures using their respective filenames (located directly using hex editor on the image) is used, as shown in Table 4. Unfortunately, there is no way to identify the present of PIM entries in the image.

File	File Signature	Foremost Result
test.htm	string "test.htm"	-
test.doc	string "test.doc"	-
test.gif	string "test.gif"	portion of the gif image extracted using Foremost, (MD5: ae8a3b9eb56b279c4fd51105c9bd092d)
test.pdf	string "test.pdf"	-
test.jpg	string "test.jpg"	-
test.mp3	string "test.mp3"	-

Table 4: File Signature

### Step 3: Acquire image after deleting files and PIM entries

Delete the six files and PIM entries in Familiar. Subsequently acquire /dev/mtd/1, and save image as `recovery_afterdelete.dd`. The image is examined using Foremost and hex editor. Table 5 shows the image analysis result. Foremost could still recover portion of the deleted gif file. The recovered gif snippet file size is different from the original gif snippet, which explains the difference in MD5 value. However, visual inspection on the retrieved gif snippet indicates the same image as the previously retrieved original gif snippet. The image is examined in hex editor to search for file signatures (using filename). For filenames with one found frequency, the offsets of the found filenames are different from the original image (`recovery_beforedelate.dd`) respective filenames' offset. For filenames with two found frequency, one of the two found offsets is similar to the original image respective filenames' offset.

File	File Signature	Found Frequency	Foremost Result
test.htm	string "test.htm"	2	-
test.doc	string "test.doc"	2	-
test.gif	string "test.gif"	2	portion of the gif image, MD5: 4d84cbcb5d43892768c0d2a813d69dc8 MD5 value different, however, visual inspection indicates the same image.
test.pdf	string "test.pdf"	1	-
test.jpg	string "test.jpg"	1	-
test.mp3	string "test.mp3"	1	-

Table 5: Deleted Files Analysis Result

### Step 4: Acquire image after soft reset

After deleting the files and PIM entries, and acquiring image, perform soft reset on Familiar. Then, acquire /dev/mtd/1 (with deleted six files and PIM entries), and saved image as `recovery_deleted_softreset.dd`. The image is examined using Foremost and hex editor. The result obtained is exactly the same as output from image `recovery_afterdelete.dd`, as shown in Table 5. The extracted gif file with the same MD5 value confirms its present. Furthermore, soft reset on the device with the deleted files does not change the deleted files offset. The offsets are still similar to the offsets in image `recovery_afterdelete.dd`.

### Step 5: Acquire image after hard reset

After soft resetting Familiar and acquiring image, perform hard reset on Familiar. Then, acquire /dev/mtd/1 image and saved as `recovery_deleted_hardreset.dd`. The image is examined using Foremost and hex editor. The result obtained is exactly the same as output from image `recovery_afterdelete.dd`, as shown in Table 5. Hard reset does not change the deleted files offset as well. The offsets are still similar to the offsets in image `recovery_afterdelete.dd`.

## Conclusion

After the files are deleted, the filenames are still present on the ROM, even after subsequent soft and hard reset. The existence of filenames could be used as potential evidence. After deleting the original gif file and followed by subsequent soft and hard resets, the original gif snippet could still be recovered. This result suggests the possibility of file recovery after actual file deletion. When a file is deleted, the actual file content is not removed. Instead only the link to the data is modified (Kruse II & Heiser, 2002). Base on deleted files' (i.e. test.htm, test.doc, test.gif) filename offsets, which is the same as the original filenames offset (before files deletion), it is possible that remnant of these three deleted files could still be found. Since JFFS2 file system features compression and without the aid of JFFS2 analysis tools, the present of these deleted files could not be confirmed.

## GARBAGE COLLECTION ANALYSIS

In the previous image analysis on deleted files, remnant of the deleted file (i.e. the gif snippet) is still present. The other possible reason deleted file is still recoverable is that garbage collection may have not been executed. When the number or free blocks reaches certain threshold, garbage collection process starts (Woodhouse, 2001). During garbage collections, the entire dirty block (block with obsolete nodes) is erased by resetting bits from zero to one. A garbage collection process should have removed all the deleted files contents.

Previous experiment on six deleted files with a total file size of 681KB, may not be sufficient enough to trigger a garbage collection, since after loading the six files into Familiar, approximately 50% of the ROM space is still free. Thus, the following test is done to further exam the possibility of garbage collection to remove remnants of deleted files.

Step 1: Acquire image on a default Familiar installation.

First, the iPAQ is loaded with a fresh default Familiar image. Subsequently, acquire `/dev/mtd/1` and saved image as `garbage_defaultinstallation.dd`. Then, analyse the acquired image using WinHex "Analyze File" feature, which will generate a report on the percentage of all ASCII characters occurrence in the image. Hexadecimal "FF" (binary 1111 1111) is used to indicate erased area.

Step 2: Load large files into ROM

To facilitate image analysis and to trigger garbage collection, three large mp3 files are used, as shown in Table 6. Subsequently, acquire `/dev/mtd/1` and save image as `garbage_beforedelate.dd`. Analyse the image for the percentage occurrence of "FF" using WinHex "Analyze File" function. The offset of all three mp3 filenames are also recorded.

Filename	File Size (kilo bytes)
garbage_testfile1.mp3	4120KB
garbage_testfile2.mp3	4115KB
garbage_testfile3.mp3	4127KB
<b>Total</b>	<b>12,362KB</b>

Table 6: Garbage Collection Test Files

Step 3: Acquire image after deleting Files

Delete the three mp3 files. Then, acquire `/dev/mtd/1`, and saved image as `garbage_afterdelete.dd`. Analyse the image for the percentage occurrence of "FF" using WinHex "Analyze File" function. Analyse the image for all three deleted mp3 filenames, and record the found offset.

Step 4: Acquire image after soft reset

After deleting the three mp3 files and acquiring image in the previous step, perform soft reset on Familiar. Then, acquire `/dev/mtd/1`, and saved image as `garbage_deleted_softreset.dd`. Analyse the image for the percentage occurrence of "FF" using WinHex "Analyze File" function. Analyse the image for all three deleted mp3 filenames, and record the found offset.

## Conclusion

Image	Description	"FF" percentage (Erased Area)	Disk Free
garbage_defaultinstallation.dd	fresh Familiar installation	54.4%	50%
garbage_beforedelate.dd	image with three mp3 files	16.14%	12%
garbage_afterdelete.dd	mp3 files deleted	48.29%	50%

garbage_deleted_softreset.dd	mp3 files deleted with soft reset	53.78%	50%
------------------------------	-----------------------------------	--------	-----

Table 7: Garbage Collection Test Result

Filename	File Signature	Found Frequency
garbage_testfile1.mp3	string "garbage_testfile1.mp3"	2
garbage_testfile2.mp3	string "garbage_testfile2.mp3"	1
garbage_testfile3s.mp3	string "garbage_testfile3.mp3"	1

Table 8: Test Filename Found Frequency

Table 7 show the results of images analysis on "FF" occurrence percentage and value of "disk free" using Unix `df` command. The values between "disk free" and "FF" percentage indicate close relation between them. Before the mp3 files are loaded into the ROM, 54.4% of the ROM space is unused. After loading the three mp3 files, only 16.14% of "FF" is left. 38.26% of the ROM space is occupied by the mp3 files. Its equivalent value in Kbytes is almost similar to total mp3 file size, 12,362KB, as shown in the below calculation.

$$32,256\text{KB (Total ROM size)} \times 0.3826 = 12,341\text{KB}$$

After deleting the mp3 files, garbage collections is believed to have been started, as the percentage of "FF" has increase dramatically. After a soft reset on the image, the percentage of "FF" in `garbage_deleted_softreset.dd` is almost similar to `garbage_defaultinstallation.dd`. It is reasonable to believe most of the mp3 file contents are erased. However, image analysis on the mp3 filenames offset shows that all the mp3 filenames are still present on all the images. Table 8 shows the found frequency of each filenames. For filenames with one found frequency, the offsets are different from the original (`garbage_beforedellete.dd`) filenames' offsets. For filename `garbage_tesfile1.mp3`, one of the two found offsets is similar to the original filename's offset.

Garbage collection process will actually remove remnant of the deleted files. PDA devices with limited storage have the tendency of executing garbage collection frequently to make space for new files. Such characteristic will minimize the possibility of deleted files recovery.

## CONCLUSION

The image integrity on Familiar iPAQ is easier to maintain compared to Pocket PC and Palm OS as Familiar stores the file system in ROM, as oppose to RAM. It is possible to maintain Familiar images integrity after consecutive acquisition, a result which is not possible on Pocket PC and Palm OS. Even though Familiar stores files in ROM, the fact that image acquisition could only be done by first login to the PDA device will cause Familiar to write login data to the ROM and hence changes the ROM integrity. Unless, direct ROM chip reading technique is used to acquire the contents, or Familiar boot loader supports OS booting from external cards, the file change is inevitable. Garbage collection in JFFS2 could remove any possible deleted file remnants. The tendency of frequent garbage collection execution on PDA devices causes difficulty in file recovery. JFFS2 implements compression, which makes file recovery using data carving technique hard to perform. Furthermore, there are no automated tools to interpret and analyse Familiar JFFS2. Hence, there is a need to develop a tool to interpret JFFS2 to facilitate forensics analysis on Familiar-based PDAs.

## REFERENCE

- Axis (2001, November 26). *JFFS - Journaling Flash File System*. Retrieved November 11, 2005, from <http://developer.axis.com/software/jffs/doc/jffs.shtml>
- Ayers, R. & Jansen W. (2004, August). *PDA Forensic Tools: An Overview and Analysis*. Retrieved November 17, 2005 from NIST: <http://csrc.nist.gov/publications/nistir/nistir-7100-PDAForensics.pdf>
- Bowman, I., Siddiqi, S., & Tanuan, M. C. (1998). *Concrete Architecture of the Linux Kernel*. Retrieved November 17, 2005 from <http://plg.uwaterloo.ca/~itbowman/CS746G/a2/>
- Brake, C. (2001, July 20). *Flash Filesystems for Embedded Linux Systems*. Retrieved November 11, 2005, <http://www.linuxjournal.com/node/4678/print>
- Carrier, B. (2005, March 17). *File System Forensic Analysis*. Addison Wesley Professional.
- Frichot, C (2004). *An Analysis of the Integrity of Palm Images Acquired with PDD*. Retrieved November 19, 2005 from Edith Cowan University: [http://scissec.scis.ecu.edu.au/publications/2004\\_FRICHOT\\_ACNIFC\\_Analysis\\_of\\_the\\_Integrity\\_of\\_Palm\\_Images\\_Acquired\\_with\\_PDD.pdf](http://scissec.scis.ecu.edu.au/publications/2004_FRICHOT_ACNIFC_Analysis_of_the_Integrity_of_Palm_Images_Acquired_with_PDD.pdf)
- Informit (2005, November 9). *PDA Forensics*. Retrieved November 19, 2005 from Informit: <http://www.informit.com/guides/printerfriendly.asp?g=security&seqNum=104>
- Jansen, W., & Ayers, R. (2004, November). *Guidelines on PDA Forensics*. Retrieved November 17, 2005 from NIST: [csrc.nist.gov/publications/nistpubs/800-72/sp800-72.pdf](http://csrc.nist.gov/publications/nistpubs/800-72/sp800-72.pdf)
- Kruse II, W.G., & Heiser, J.G. (2002). *Computer Forensics: Incident Response Essentials*. Boston: Addison Wesley.
- Grant, J. (2002). *Pdd: Memory Imaging and Forensic Analysis of Palm OS Devices*. Retrieved November 21, 2005 from [http://www.grandideastudio.com/files/security/mobile/pdd\\_palm\\_forensics.pdf](http://www.grandideastudio.com/files/security/mobile/pdd_palm_forensics.pdf)
- Hillerman, G. (2003). *Palm OS File Format Specification*. Retrieved November 21, 2005 from PalmSource, Inc: <http://www.palmos.com/dev/support/docs/fileformats/FileFormatsTOC.html>
- Intel (2005, September). *Intel® Persistent Storage Manager (Intel® PSM) User's Guide, Version 3.8*. Retrieved November 12, 2005, from <http://download.intel.com/design/flcomp/manuals/29813610.pdf>
- Malik, V. (2001, August 13). *The Linux MTD, JFFS HOWTO*. Retrieved November 12, 2005, from <http://ftp.linux.org.uk/pub/people/dwmw2/mtd/cvs/mtd/mtd-jffs-HOWTO.txt>
- Mislan, R.P. (n.d.). *Acquisition of a Palm OS PDA using @Stake's Palm dd, Paraben's PDA Seizure, and Guidance Software's EnCase*. Retrieved November 21, 2005 from <http://www.htcia-mountainstates.org/palmosacquisition.pdf>
- PalmSource (n.d.). *Palm OS® Programmer's Companion Volume I: 5 Memory*. Retrieved November 21, 2005 from <http://www.palmos.com/dev/support/docs/palmos/Memory.html>
- PaulEggleton (2005, August 12). *FamiliarFaq*. Retrieved November 17, 2005 from <http://handhelds.org/moin/moin.cgi/FamiliarFaq#head-87a7fac0185ca2be60ecd1a946827adef5921208>
- TobyGray (n.d.). *Familiar Backup Howto - Handhelds.org - Open source for handheld devices*. Retrieved November 21, 2005, from <http://handhelds.org/moin/moin.cgi/FamiliarBackupHowto>
- Wilson, G. (2004, January 28). *Exploring Palm OS®*. Retrieved November 21, 2005 from [http://www.palmos.com/dev/support/docs/protein\\_books/memory\\_databases\\_files.pdf](http://www.palmos.com/dev/support/docs/protein_books/memory_databases_files.pdf)
- Woodhouse, D. (2001). *JFFS : The Journaling Flash File System*. Retrieved November 10, 2005, from <http://sources.redhat.com/jffs2/jffs2.pdf>
- Valli, C. (2005). *Issues Relating to the Forensics Analysis of PDA and Telephony (PDAT) Enabled Devices*.
- Yaghmour, K. (2003). *Building Embedded Linux Systems*. United States of America: O'Reilly.