Paradigm Solutions

Extended FAT File System

exFAT

Jeff Hamm

09

Plan. Perform. Assure.

Table of Contents

Abstract

The work herein is an excerpt from a Microsoft File Systems class being developed by Jeff Hamm for Paradigm Solutions and its clients. Many examiners have had exposure to the FAT and NTFS file systems, but few have had training on Microsoft's newest file system, Extended FAT (exFAT). This information is provided as a base line to showcase the file system and explain the significance it will have in the computer forensic community.

The material provided was designed as a three hour presentation to be used in conjunction with progressive live demonstrations. Time constraints will limit – and may exclude – the ability to effectively provide a series of live demonstrations of the file system.

At the time of this writing, AccessData and Guidance Software – the developers of the two highest profile forensic tools – do not have their forensic suites capable of viewing exFAT logically. The tools will view the file system as an unallocated area or as free space. WinHex is able to recognize the file system but will not display the logical folder structure. Because of the efficiency of exFAT in maintaining contiguous files whenever possible, the examiner will have luck in retrieving file artifacts by data carving on an exFAT drive. A need to show intent to possess, for a time line of a file, or for any other file system metadata to be presented as evidence will require an examiner to rebuild these file systems manually (for now).

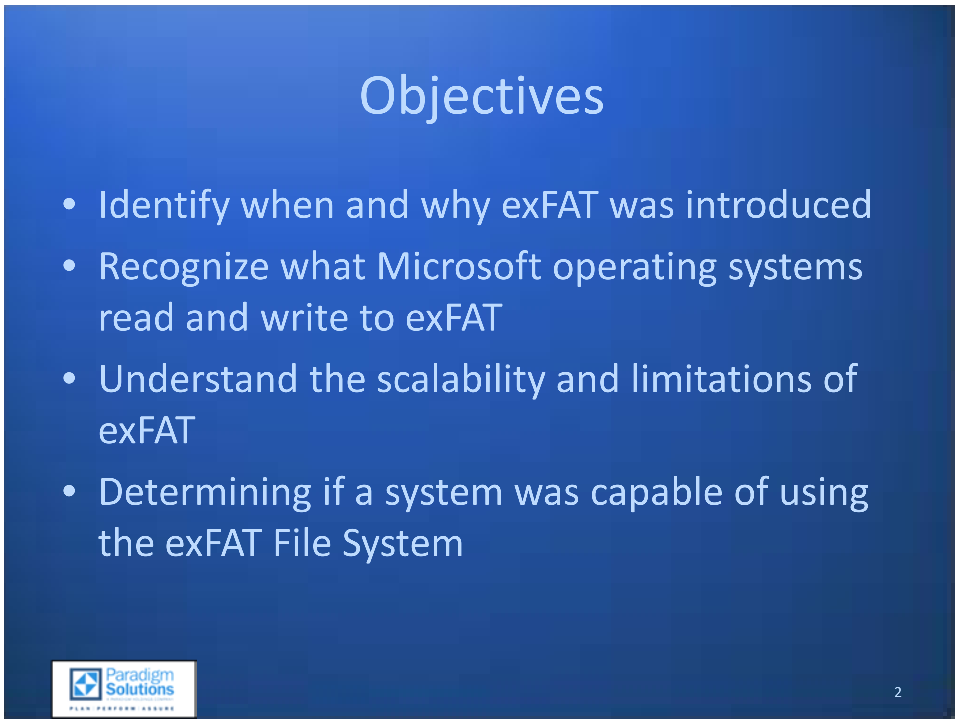
The presentation and this paper begins with the history of the file system, discusses the reasons the examiner needs to be aware of the file system, details the forensic implications, and finally examines the technical details of the file system. An appendix containing the compiled tables for locating data manually on the file system is also attached. These tables are designed to be a quick reference resource for an examiner.

Much of the information was ascertained through research and testing by the author with additional research conducted by fellow examiners working alongside the author. The Microsoft patent application for the file system was an invaluable tool to validate research and theories. Finally, the independent research conducted by Jared Myers of the DCFL was critical in developing this material in the most complete and accurate fashion.

Senior Computer Forensic Analyst
Jeffery Hamm, CFCE, ACE
Paradigm Solutions
Contractor for the US Department of State
Computer Investigations and Forensics
9715 Key West Avenue, Third Floor
Rockville, Maryland 20850
hammjd@yahoo.com

exFAT File System

The History of exFAT



Objectives

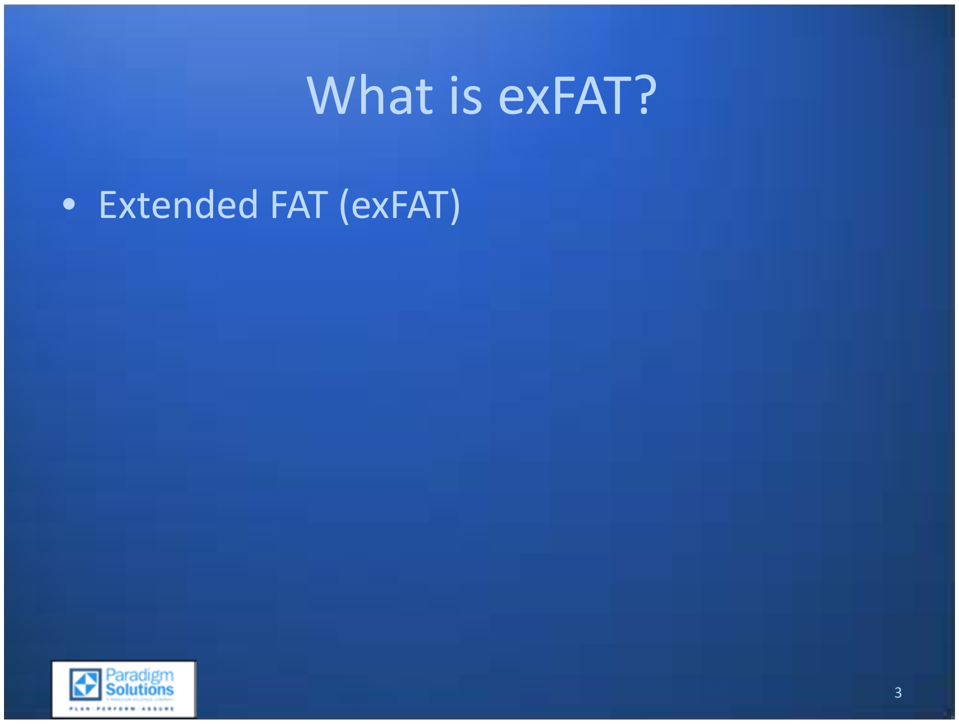
- Identify when and why exFAT was introduced
- Recognize what Microsoft operating systems read and write to exFAT
- Understand the scalability and limitations of exFAT
- Determining if a system was capable of using the exFAT File System

2

Objectives

With regards to the new exFAT (Extended FAT) file system, this module will provide the examiner with the following abilities:

- Identify when and why exFAT was introduced
- Recognize which Microsoft operating systems read and write to exFAT
- Understand the scalability and limitations of exFAT
- Determining whether a system was capable of using the exFAT File System



What is exFAT?

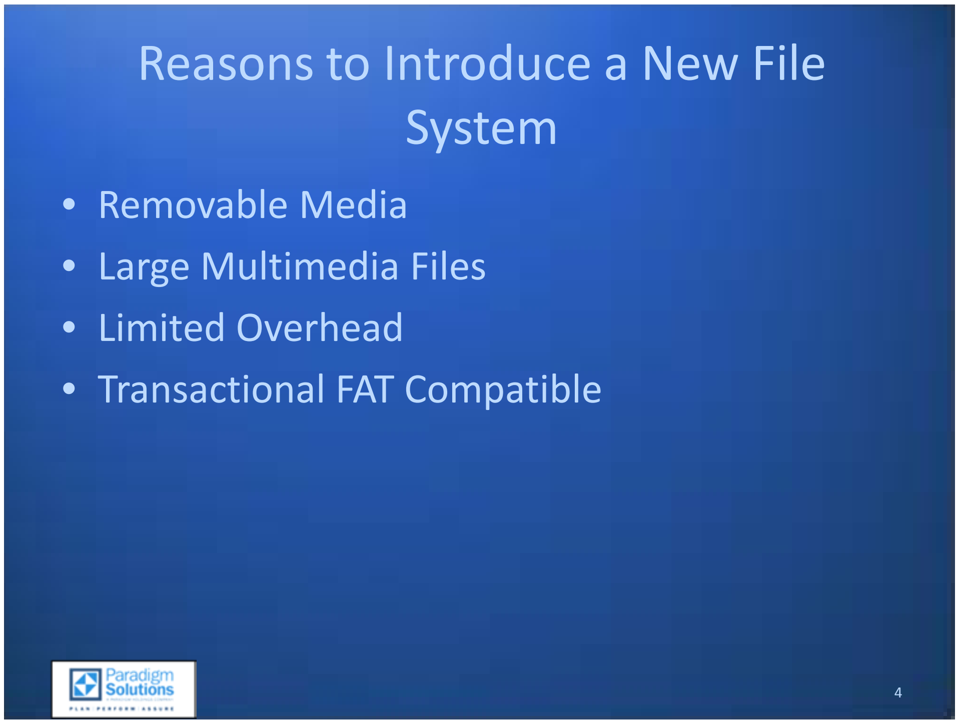
- Extended FAT (exFAT)

3

What is exFAT?

The exFAT file system was quietly introduced by Microsoft with Windows CE in 2006. It was also included with the Windows Vista SP1 update in the summer of 2008, which by default formatted new removable media as exFAT. This new file system is not recognized by most forensic tools as a logical file system even at the time of this writing (September 2009). Furthermore, in Windows XP, if drivers have not been installed beforehand, the OS will prompt a user who has inserted a device formatted in exFAT to format the disk before it can be used.

The SD Card Association introduced the SDXC standard in January of 2009. This SD card will be utilized to full capacity with native host machines (cell phones, digital video, cameras). The hosts will only recognize exFAT. (SD Association)



Reasons to Introduce a New File System

- Removable Media
- Large Multimedia Files
- Limited Overhead
- Transactional FAT Compatible

4

Reasons to Introduce a New File System

To understand the reasons behind the desire to create a new file system requires a quick review of the two primary file systems released by Microsoft.

The FAT file system is a reliable file system released by Microsoft in the early days of computing. The system is simple in the sense that it tracks data fragmentation through a File Allocation Table (FAT), while all other metadata is tracked with a directory entry (starting cluster, file name, file size, file system attributes etc). FAT has several incarnations: FAT12 is still used with floppy disks today, FAT16 expanded the file system addressing to 16 bits and was used on hard disks, and FAT32 increased addressing to 32 bits. The 32 bit addressing allows a maximum file size of $2^{32} – 1$ byte, which is one byte less than 4 GB.

File sizes have grown dramatically with the introduction of digital recording and imaging devices. Many of these devices record digital information onto removable media or flash memory. A standard DVD video file can by over 4.7 GB alone. The FAT file system, then, is less than ideal, saying nothing about the limitation on the number of entries that can be held in a FAT directory.

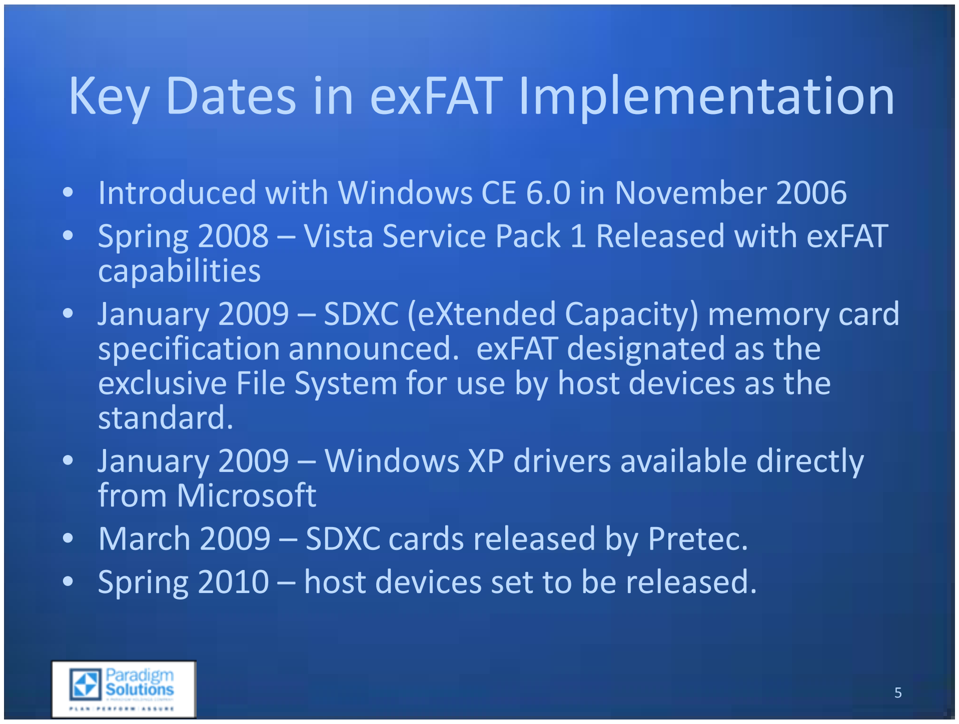
NTFS was released with Windows NT and included numerous security and redundancy features. The NTFS file system uses metadata system files to track files, metadata, and security features. These files create a significant amount of overhead on a device, as they can easily take up 56KB of data area or more after an initial format. These metadata system files then grow in size as files are added to the volume. Furthermore, NTFS was designed as a “lazy write” file system. This allowed the file system to become corrupted when a removable device is unplugged without first ejecting the device.

With size at a premium and with the potential for corrupted data, NTFS is not an ideal solution for a removable device. A simple format test on a 6 GB flash drive indicated that without adding a single user file, the metadata system files took up 56 KB of space – not significant at first glance, but as user created files are added, the metadata system files will grow rapidly.

Because of the current file system limitations and overhead, and need to protect data from unintentional ejects or devices being powered off, a new file system was seen as necessary to address these concerns. exFAT was the Microsoft solution.

Transactional FAT, or TFAT, ensures that changes to the FAT occur first in the backup FAT (FAT1) and are not committed to the FAT (FAT0) until the operation is completed. exFAT is scaled to be TFAT compatible. In the case of exFAT, as will be later discussed, the traditional FAT has actually been replaced by a Linked List, which itself has a backup. While it is not currently enabled in Vista, it is designed to be compatible with the TFAT module. In addition, TexFAT (Transactional exFAT) will implement a transaction-safe means of writing to the root directory by storing a backup of the root directory as a subdirectory, only committing changes to the root after they are completed in the backup root.

Other considerations taken in the development of exFAT include Universal Time Code (UTC) support, better flow of contiguous data for single files, and a cluster map for quicker allocation of a file to the volume.



Key Dates in exFAT Implementation

- Introduced with Windows CE 6.0 in November 2006
- Spring 2008 – Vista Service Pack 1 Released with exFAT capabilities
- January 2009 – SDXC (eXtended Capacity) memory card specification announced. exFAT designated as the exclusive File System for use by host devices as the standard.
- January 2009 – Windows XP drivers available directly from Microsoft
- March 2009 – SDXC cards released by Pretec.
- Spring 2010 – host devices set to be released.

5

Key Dates in exFAT Implementation

Windows CE (Embedded Compact) was designed for use on small devices such as mini computers, advanced cell phones, GPS units, and even robotic devices. With the release of CE 6.0 in November 2006, exFAT was released as a supported file system.

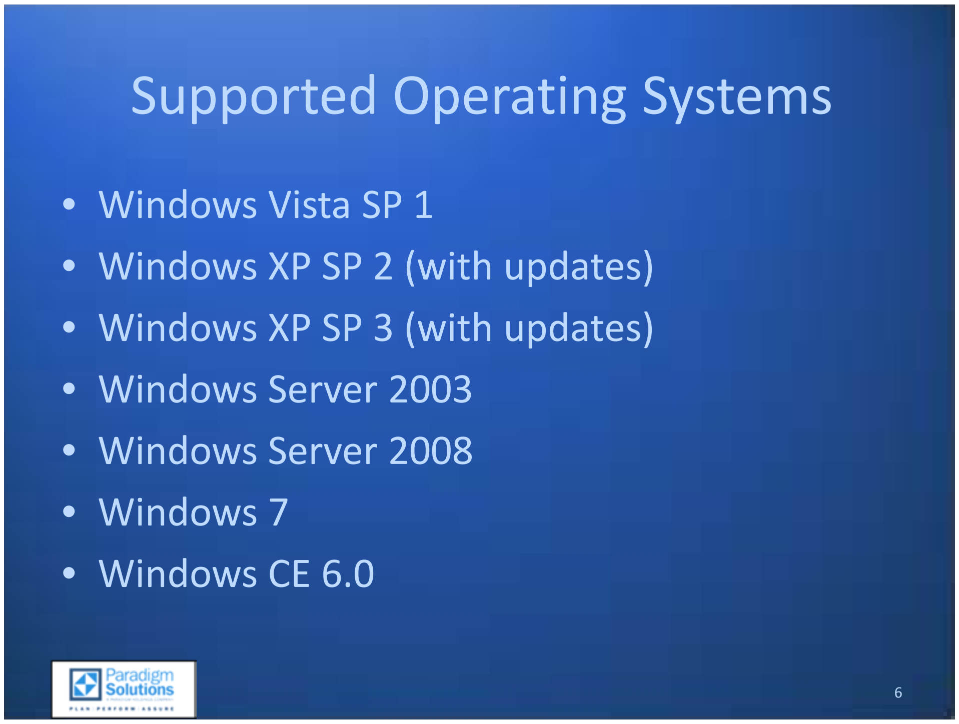
Mainstream OS support came in the spring of 2008 with the release of Windows Vista SP1 (Service Pack 1). Vista SP1 set exFAT as the default file system when formatting removable devices.

In January 2009, the SD card standard (www.sdcard.com) announced the new specifications for a new SD format: SDXC (SD eXtended Capacity). This standard included the exclusive use of exFAT as the default file system choice. SDXC allows the theoretical capacity of up to 2 TB, rendering the standard FAT file system insufficient for use on the new media cards.

Also in January 2009, Microsoft released exFAT drivers for Windows XP. The drivers require either Windows XP SP2 or SP3. They can be downloaded from http://www.microsoft.com/downloads.

March 2009 saw the release of the first SDXC cards by Pretec. Toshiba has since released SDXC cards, with other manufacturers sure to follow (at the time of this writing Panasonic was planning to release a 64 GB SDXC card). The cards are backwards compatible with other SD standards, though the volume size limitation will limit larger sized cards from being fully utilized.

The first devices capable of full SDXC utilization are scheduled to be released in the spring of 2010. It’s still unknown which device and which manufacturer will be first to market.



Supported Operating Systems

- Windows Vista SP 1
- Windows XP SP 2 (with updates)
- Windows XP SP 3 (with updates)
- Windows Server 2003
- Windows Server 2008
- Windows 7
- Windows CE 6.0

6

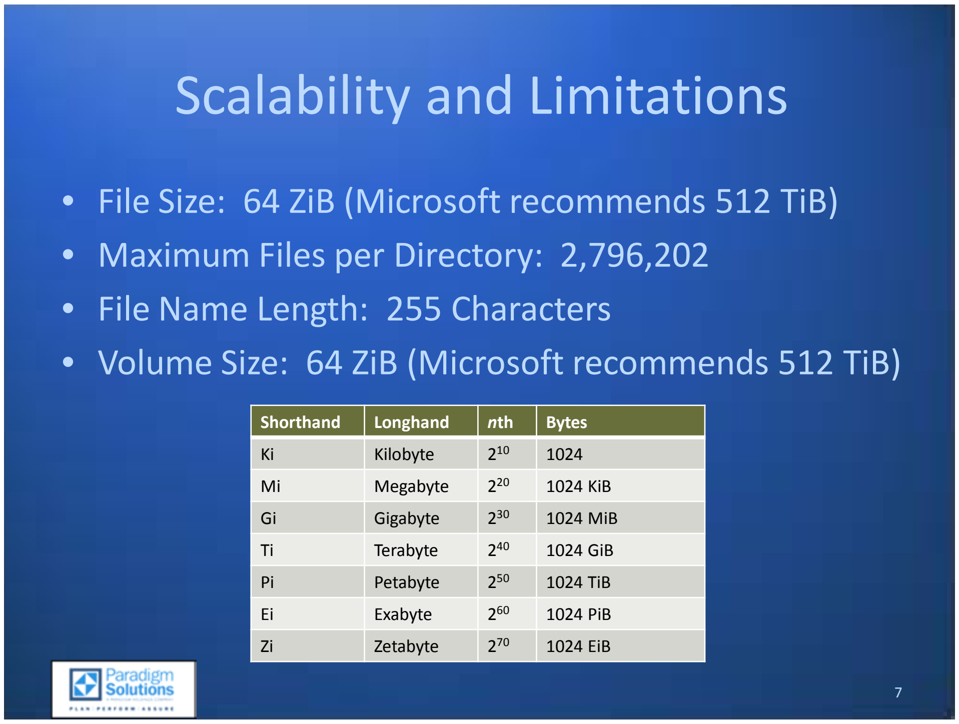
Supported Operating Systems

The only operating systems that will read from and write to exFAT are:

- Windows Vista SP1
- Windows XP SP2 (with manual updates)
- Windows XP SP3 (with manual updates)
- Windows Server 2003 (with manual updates)
- Windows Server 2008 (with manual updates)
- Windows 7
- Windows CE 6.0

To utilize the exFAT file system, Vista needs to have SP1 or higher installed. exFAT compatible drivers are available for Windows XP SP2 and SP3, but these are not installed through automatic updates and instead must be downloaded manually from Microsoft's site. If an operating system has not been updated, it will only see the physical device, not the file system, and will offer to format the media for the user.

UNIX and Linux are not currently capable of reading or writing to exFAT devices. An agreement to develop Linux exFAT drivers was sealed between Tuxera and Microsoft. It's not clear when these drivers will be released and how they will be distributed, as the exFAT file system is a closed-source, patented file system.



Scalability and Limitations

- File Size: 64 ZiB (Microsoft recommends 512 TiB)
- Maximum Files per Directory: 2,796,202
- File Name Length: 255 Characters
- Volume Size: 64 ZiB (Microsoft recommends 512 TiB)

Shorthand	Longhand	*nth*	Bytes
Ki	Kilobyte	2^{10}	1024
Mi	Megabyte	2^{20}	1024 KiB
Gi	Gigabyte	2^{30}	1024 MiB
Ti	Terabyte	2^{40}	1024 GiB
Pi	Petabyte	2^{50}	1024 TiB
Ei	Exabyte	2^{60}	1024 PiB
Zi	Zetabyte	2^{70}	1024 EiB

7

Scalability and Limitations

Limitations include:

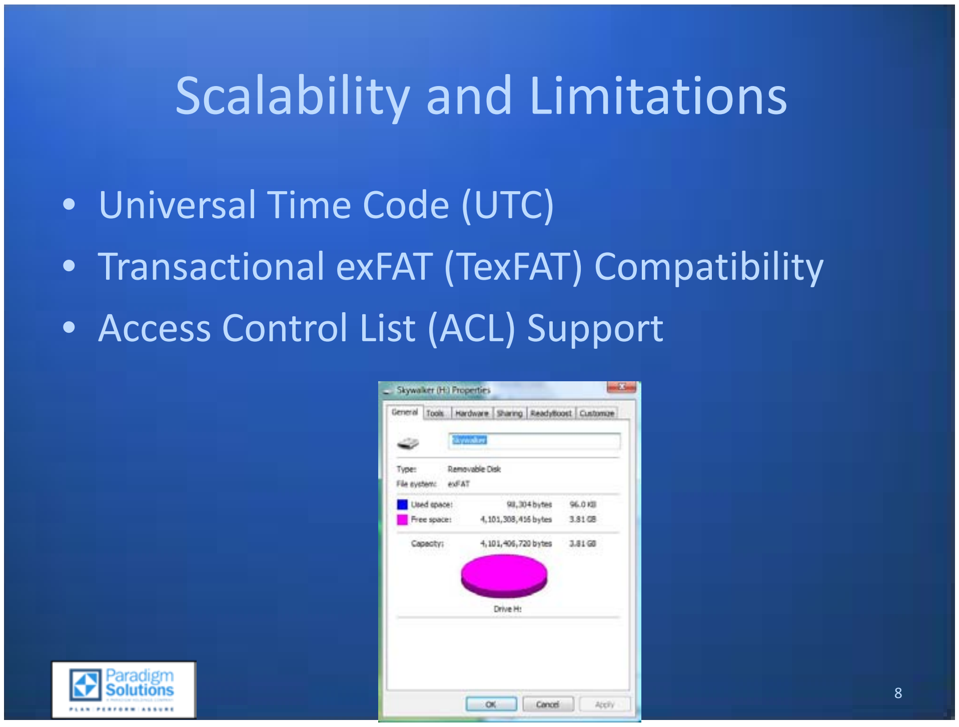
- File Size: 64 Zetabytes (recommended limitation of 512 Terabytes)
- Maximum Files per Directory: 2,796,202
- File Name Length: 255 Characters
- Volume Size: 64 Zetabytes (recommended limitation of 512 Terabytes)

Although some writings on the web refer to exFAT as FAT64, this is not an accurate description of what exFAT really is. Unlike the previous revisions of FAT (FAT12, FAT16, FAT32), exFAT was built from the ground up, focusing on scalability and flexibility in an uncertain future. FAT64 would imply a maximum file size of 2^{64}, which would be 16 EB (Exabytes). Microsoft's documentation on exFAT indicates that the maximum file size is 64 Zetabytes, which defies the 64 bit addressing limitation. Documentation on Wikipedia indicates maximum file size is 127 PB (Petabytes) without defining what causes that limitation to occur. (Wikipedia, 2009) and (Microsoft)

A list of relative measurement units is provided in Table 1 - Data Measurement Units.

Shorthand	Longhand	*n*th	Bytes
Ki	Kilobyte	2^{10}	1024
Mi	Megabyte	2^{20}	1024 KiB
Gi	Gigabyte	2^{30}	1024 MiB
Ti	Terabyte	2^{40}	1024 GiB
Pi	Petabyte	2^{50}	1024 TiB
Ei	Exabyte	2^{60}	1024 PiB
Zi	Zetabyte	2^{70}	1024 EiB

Table 1 - Data Measurement Units



Scalability and Limitations

- Universal Time Code (UTC)
- Transactional exFAT (TexFAT) Compatibility
- Access Control List (ACL) Support

Skywalker (H:) Properties
General
Tools
Hardware
Sharing
ReadyBoost
Customize
Skywalker
OK
Cancel
Apply

Type: Removable Disk
File system: exFAT

🟦	Used space:	98,304 bytes	96.0 KB
🟪	Free space:	4,101,308,416 bytes	3.81 GB
	Capacity:	4,101,406,720 bytes	3.81 GB

Drive H:
8

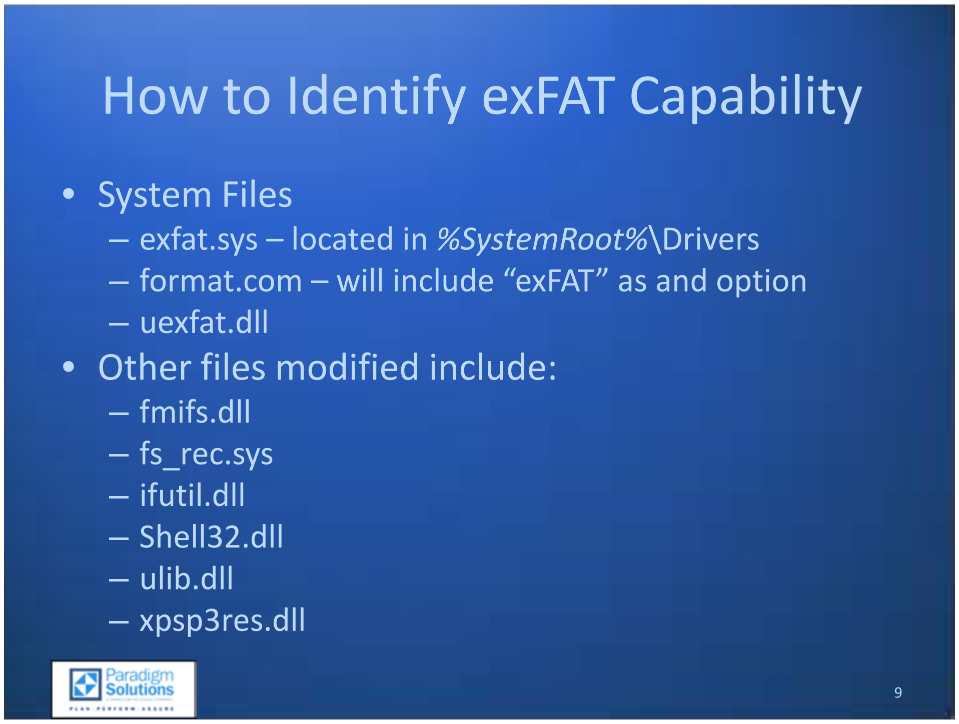
Scalability and Limitations Continued

In addition to addressing size limitations, exFAT has introduced compatibility for advanced features. Dates and times in the file system are now stored in UTC and are based on the start of the previous Epoch, January 1, 1980. This allows date resolution to be recorded to the 10th of a millisecond in a 64 bit Windows date and time stamp.

Transactional exFAT (TexFAT), based on TFAT, allows recovery from critical failure. TexFAT is not the same as journaling. Rather, it uses the FAT1 and a backup root directory to store changes until the write operation has completed, at which point it will synchronize with FAT0 and/or the actual root directory[1]. Microsoft documentation claims that the backup root directory exists as a subdirectory in the root. Because of overhead, TexFAT limits a file name to 247 characters; attempting to restore a corrupted file with more than 247 characters will be met with failure. (Microsoft)

Access Control List (ACL) support has not been currently implemented on exFAT, but Microsoft has indicated they plan to pursue this functionality. ACLs limit access permissions for a file or directory based on user and/or group status.

[1] An actual backup Linked List and a backup Root Directory have not been located by the author. It is not clear that TexFAT has been implemented to date – and certainly is not implemented in the authors Vista and XP systems.



How to Identify exFAT Capability

- System Files
  - exfat.sys – located in *%SystemRoot%*\Drivers
  - format.com – will include “exFAT” as and option
  - uexfat.dll
- Other files modified include:
  - fmifs.dll
  - fs_rec.sys
  - ifutil.dll
  - Shell32.dll
  - ulib.dll
  - xpsp3res.dll

9

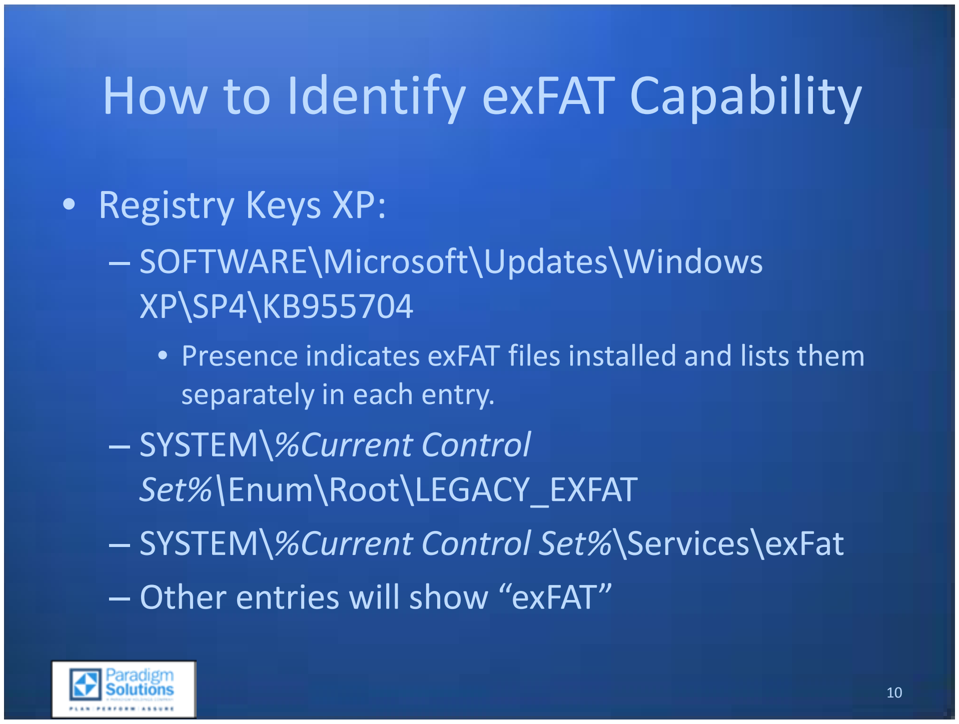
How to Identify exFAT Capability – Required Files

Three files will be present when a non-native capable operating system is updated to include exFAT support: exfat.sys, format.com, and uexfat.dll. exfat.sys and uexfat.dll are new files that will be located in the *%SystemRoot%*\System32\Drivers\ and the *%SystemRoot%*\System32\ folder respectively. format.com is updated to include “exFAT” as a formatting option.

Additional modified files include:

- fmifs.dll
- fs_rec.sys
- ifutil.dll
- Shell32.dll
- ulib.dll
- xpsp3res.dll

The lack of these files would indicate that the system would be unable to read an exFAT volume. (Microsoft)



How to Identify exFAT Capability

- Registry Keys XP:
  - SOFTWARE\Microsoft\Updates\Windows XP\SP4\KB955704
    - Presence indicates exFAT files installed and lists them separately in each entry.
  - SYSTEM\*%Current Control Set%*\Enum\Root\LEGACY_EXFAT
  - SYSTEM\*%Current Control Set%*\Services\exFat
  - Other entries will show “exFAT”

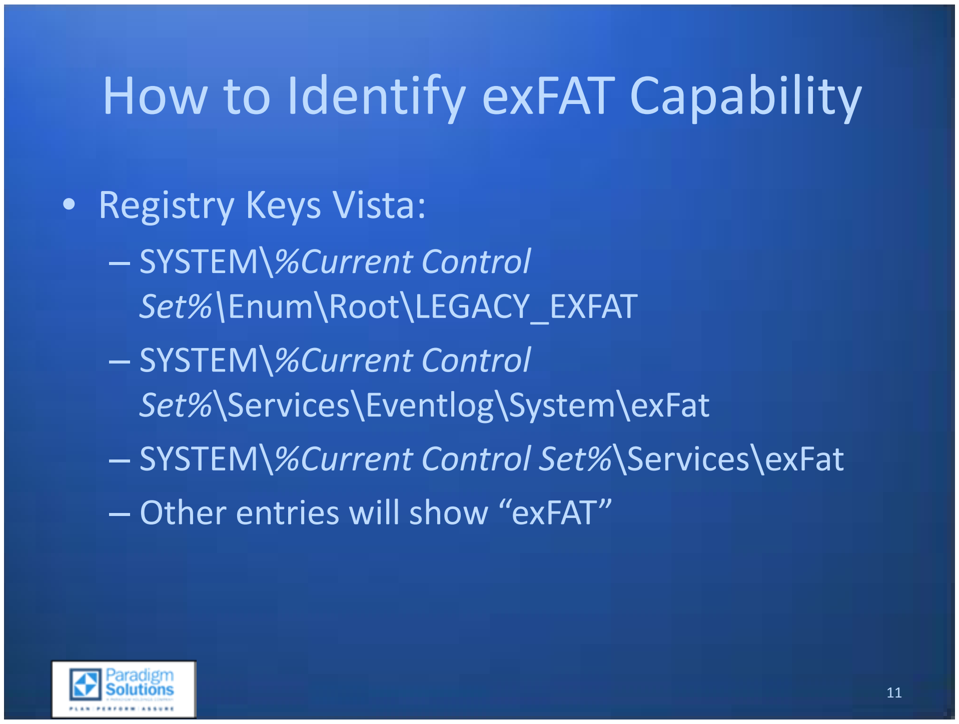
10

How to Identify exFAT Capability – XP Registry Keys

When conducting an exam including a piece of exFAT-formatted media, the examiner should take steps to ensure that the suspect system had the capability to read/write to the exFAT file system. The registry keys to verify operability on a Windows XP machine include (but are not limited to) the following:

- SOFTWARE\Microsoft\Updates\Windows XP\SP4\KB955704
- SYSTEM\%Current Control Set%\Enum\Root\LEGACY_EXFAT
- SYSTEM\%Current Control Set%\Services\exFat

Additional instances of “exFAT” will be present in a search of a registry with exFAT service installed.



How to Identify exFAT Capability

- Registry Keys Vista:
  - SYSTEM\*%Current Control Set%*\Enum\Root\LEGACY_EXFAT
  - SYSTEM\*%Current Control Set%*\Services\Eventlog\System\exFat
  - SYSTEM\*%Current Control Set%*\Services\exFat
  - Other entries will show "exFAT"

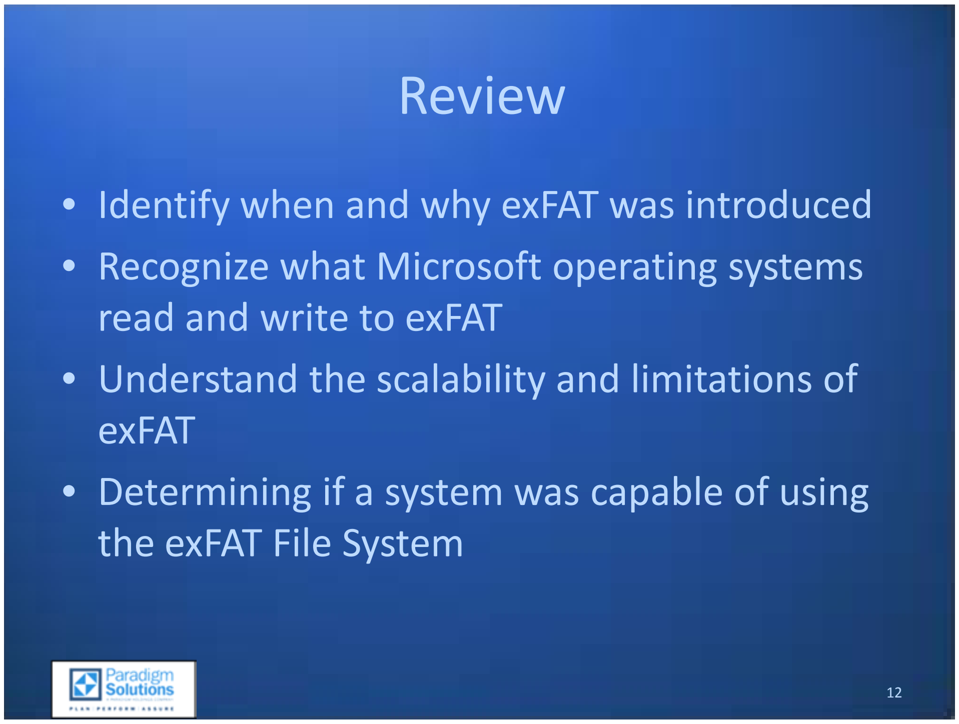
11

How to Identify exFAT Capability – Vista Registry Keys

The Windows Vista Registry keys that register the exFAT services are slightly different and are listed below. Not mentioned here is Windows 7, as Windows 7 by default has the capability of reading and writing to exFAT.

- SYSTEM\%Current Control Set%\Enum\Root\LEGACY_EXFAT
- SYSTEM\%Current Control Set%\Services\Eventlog\System\exFat
- SYSTEM\%Current Control Set%\Services\exFat
- Other entries will show "exFAT"

Additional instances of "exFAT" will be present in a search of a registry with exFAT service installed.



Review

- Identify when and why exFAT was introduced
- Recognize what Microsoft operating systems read and write to exFAT
- Understand the scalability and limitations of exFAT
- Determining if a system was capable of using the exFAT File System

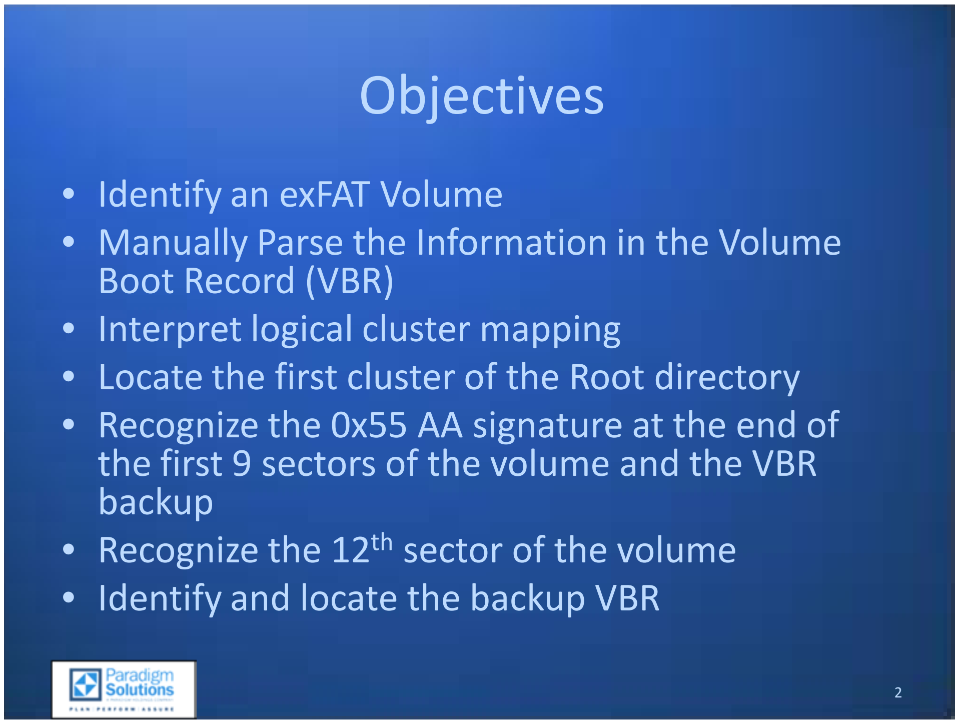
12

Review

With the completion of this section, review how to do the following:

- Identify when and why exFAT was introduced
- Recognize what Microsoft operating systems read and write to exFAT
- Understand the scalability and limitations of exFAT
- Determine whether a system was capable of using the exFAT File System

Volume Boot Record



Objectives

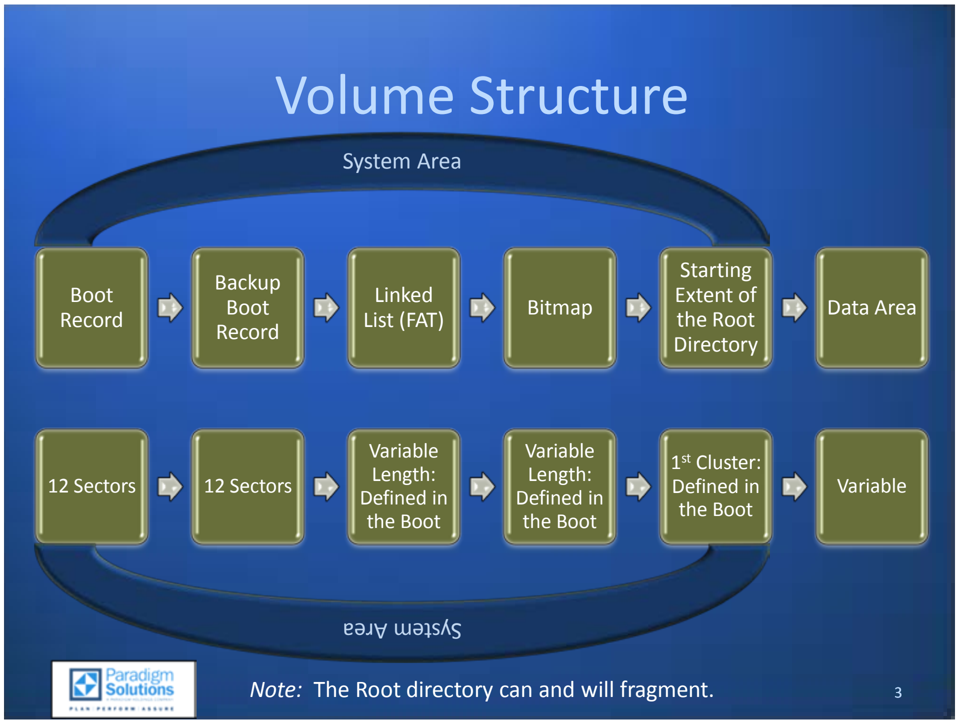
- Identify an exFAT Volume
- Manually Parse the Information in the Volume Boot Record (VBR)
- Interpret logical cluster mapping
- Locate the first cluster of the Root directory
- Recognize the 0x55 AA signature at the end of the first 9 sectors of the volume and the VBR backup
- Recognize the 12th sector of the volume
- Identify and locate the backup VBR

2

Objectives

To examine an exFAT volume, it is critical to recognize the file system as exFAT. Data in the volume boot record (VBR) define the on-disk structures used by exFAT. This portion will give the examiner the following skills and abilities:

- Identify an exFAT Volume
- Manually Parse the Information in the Volume Boot Record (VBR)
- Interpret logical cluster mapping
- Locate the first cluster of the Root directory
- Recognize the 0x55 AA signature at the end of the first 9 sectors of the volume and the VBR backup
- Recognize the 12th sector of the volume
- Identify and locate the backup VBR



Volume Structure

System Area
Boot Record
Backup Boot Record
Linked List (FAT)
Bitmap
Starting Extent of the Root Directory
Data Area
12 Sectors
12 Sectors
Variable Length: Defined in the Boot
Variable Length: Defined in the Boot
1st Cluster: Defined in the Boot
Variable
System Area

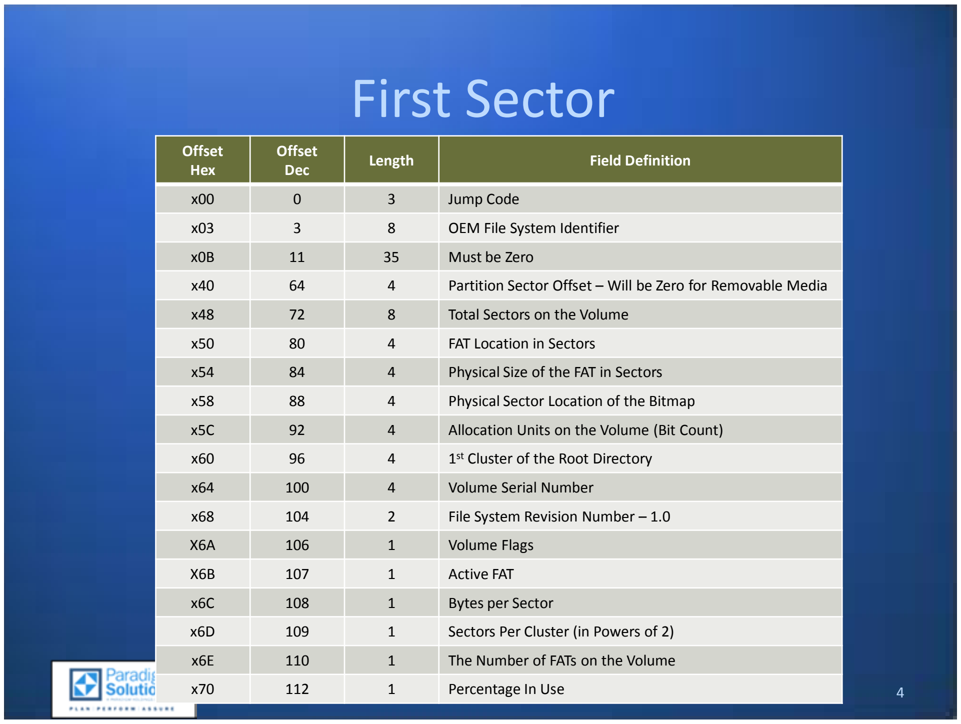
Note: The Root directory can and will fragment.

3

Volume Structure

The volume structure of exFAT is similar to that of FAT32. A fixed System Area at the start of the volume will contain the Volume Boot Record (VBR), a Backup VBR, the Linked List (or FAT), the Volume Bitmap, and the starting extent of the Root Directory. The data area will exist outside of this system area.

The VBR and backup VBR will be explained in depth in this portion of the document. The Linked List tracks the fragmentation of a file, like a more robust traditional FAT. The Bitmap tracks the allocation status of every addressable unit on the volume. Finally, the Root Directory will have a static first cluster as defined in the VBR. The root directory can and will fragment based on the needs of the file system.



First Sector

Offset Hex	Offset Dec	Length	Field Definition
x00	0	3	Jump Code
x03	3	8	OEM File System Identifier
x0B	11	35	Must be Zero
x40	64	4	Partition Sector Offset – Will be Zero for Removable Media
x48	72	8	Total Sectors on the Volume
x50	80	4	FAT Location in Sectors
x54	84	4	Physical Size of the FAT in Sectors
x58	88	4	Physical Sector Location of the Bitmap
x5C	92	4	Allocation Units on the Volume (Bit Count)
x60	96	4	1st Cluster of the Root Directory
x64	100	4	Volume Serial Number
x68	104	2	File System Revision Number – 1.0
X6A	106	1	Volume Flags
X6B	107	1	Active FAT
x6C	108	1	Bytes per Sector
x6D	109	1	Sectors Per Cluster (in Powers of 2)
x6E	110	1	The Number of FATs on the Volume
x70	112	1	Percentage In Use

4

First Sector Offset Table

Located at the first sector of the Volume, the Volume Boot Record will designate specific locations for various data artifacts. This data is broken down into a table for quick reference. Locating the data involves navigating to the correct offset and interpreting the values, as most of the forensically relevant data is a simple value stored by the file system. This information was based on testing, personal observations in behavior by the author, and Microsoft's exFAT patent (Pudipeddi, Ghotge, & Thind, 2009). As with previous Microsoft file systems, all addressing data should be interpreted as Little Endian unless otherwise noted. The VBR will contain the following:

- The Jump Code, which must be present in a Microsoft file system,
- The OEM File System Identifier, which in this case is for exFAT,
- The total sectors on the volume,
- The Partition Sector Offset, which will be zero for removable media or the volume location for a hard disk,
- The sector location of the first FAT,
- The physical size of the FAT, which is the number of sectors that make up the FAT,
- The physical sector location of the bitmap, which tracks the allocation status on the volume,
- The allocation units on the volume, which is the number of bits used to track allocation, also called the bit count,
- The logical location of the first cluster of the root directory,
- The volume serial number, which was created at format.

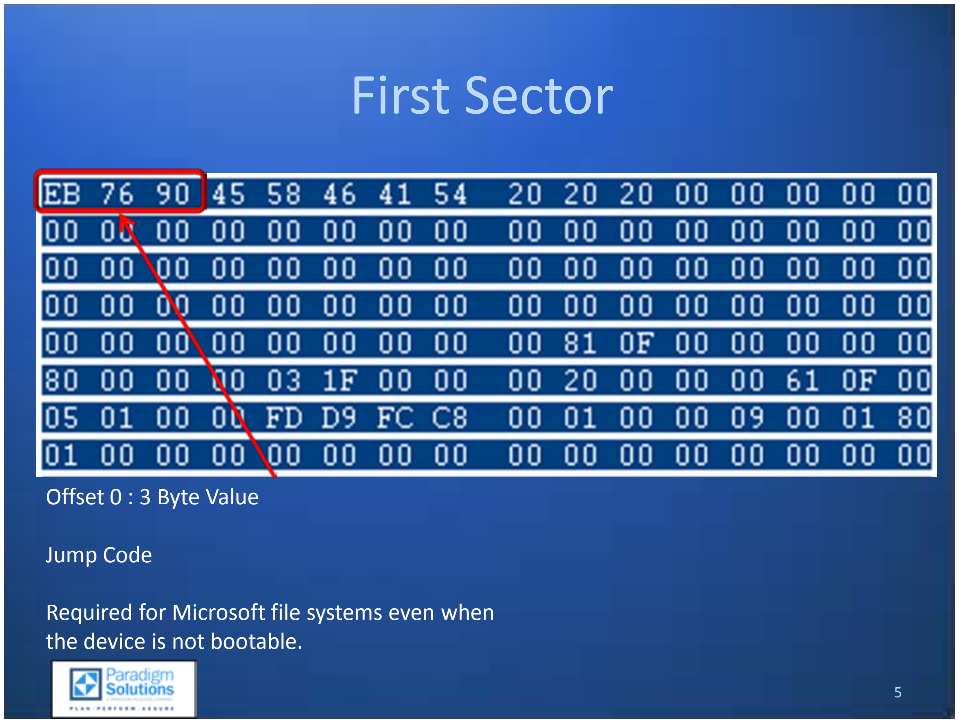
- The file system revision number, currently 1.0, which is stored with the high byte as the major system revision number and the low byte as the minor system revision number (0x00 01),
- Volume Flags, which track which Linked List and which bitmap are active, as well as the volume state. This has not been tested thoroughly at the time of this writing; the information from the Flags has been derived primarily from the Microsoft Patent for the exFAT file system. The values are stored as individual bits: 0x01 – volume dirty, 0x02 – media failure, 0x04 – clear to zero 0x08-FF – reserved,
- The Active FAT, which describes which FAT and bitmap are active, with a value of 0 to indicate the first FAT and bitmap and a value of 1 to indicate the second,
- The sectors per cluster ratio, calculated as 2 to the power of the value (i.e. a value of 2 here would indicate 2^2 sectors per cluster, meaning 4 sectors per cluster, meaning or 2048 bytes per cluster), and
- The Partition Sector Offset, which will only be on hard disks with a Master Boot Record. On removable media, this value will be zero .

A complete list of known values is provided in Table 2 - exFAT Volume Boot Record

.

Offset Hex	Offset Dec	Length	Field Definition
x00	0	3	Jump Code
x03	3	8	OEM File System Identifier
x0B	11	35	Must be Zero
x40	64	4	Partition Sector Offset – Will be Zero for Removable Media
x48	72	8	Total Sectors on the Volume
x50	80	4	FAT Location in Sectors
x54	84	4	Physical Size of the FAT in Sectors
x58	88	4	Physical Sector Location of the Bitmap
x5C	92	4	Allocation Units on the Volume (Bit Count)
x60	96	4	1st Cluster of the Root Directory
x64	100	4	Volume Serial Number
x68	104	2	File System Revision Number – 1.0
X6A	106	1	Volume Flags
X6B	107	1	Active FAT
x6C	108	1	Bytes per Sector
x6D	109	1	Sectors Per Cluster (in Powers of 2)
x6E	110	1	The Number of FATs on the Volume
x70	112	1	Percentage In Use

Table 2 - exFAT Volume Boot Record



First Sector

```
EB 76 90 45 58 46 41 54  20 20 20 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 81 0F 00 00 00 00 00
80 00 00 00 03 1F 00 00  00 20 00 00 00 61 0F 00
05 01 00 00 FD D9 FC C8  00 01 00 00 09 00 01 80
01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

Offset 0 : 3 Byte Value

Jump Code

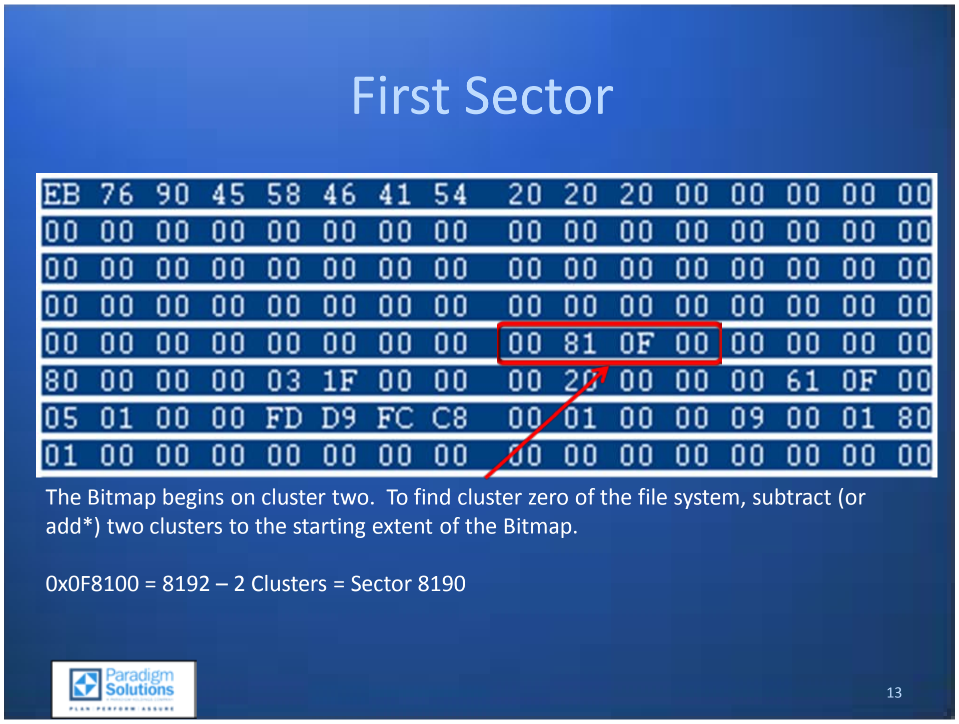
Required for Microsoft file systems even when the device is not bootable.

5

Locating the First Sector Offsets

Traditionally, the Volume Boot Record is considered the first sector of a boot device. With exFAT, the traditional values for cluster size, sector count, file system starting location, and more still reside in the first sector of the volume. The boot record for exFAT also includes file system specific artifacts for the first 24 sectors of the volume. The additional information stored in them will be discussed later.

The information contained in the first sector can be reviewed by looking at the Volume Boot Record table provided. Each offset in the record contains data specific to the file system. Some values are as yet unknown. All offsets and the data represented were found through research and personal observations.



First Sector

```
EB 76 90 45 58 46 41 54  20 20 20 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 81 0F 00 00 00 00 00
80 00 00 00 03 1F 00 00  00 20 00 00 00 61 0F 00
05 01 00 00 FD D9 FC C8  00 01 00 00 09 00 01 80
01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

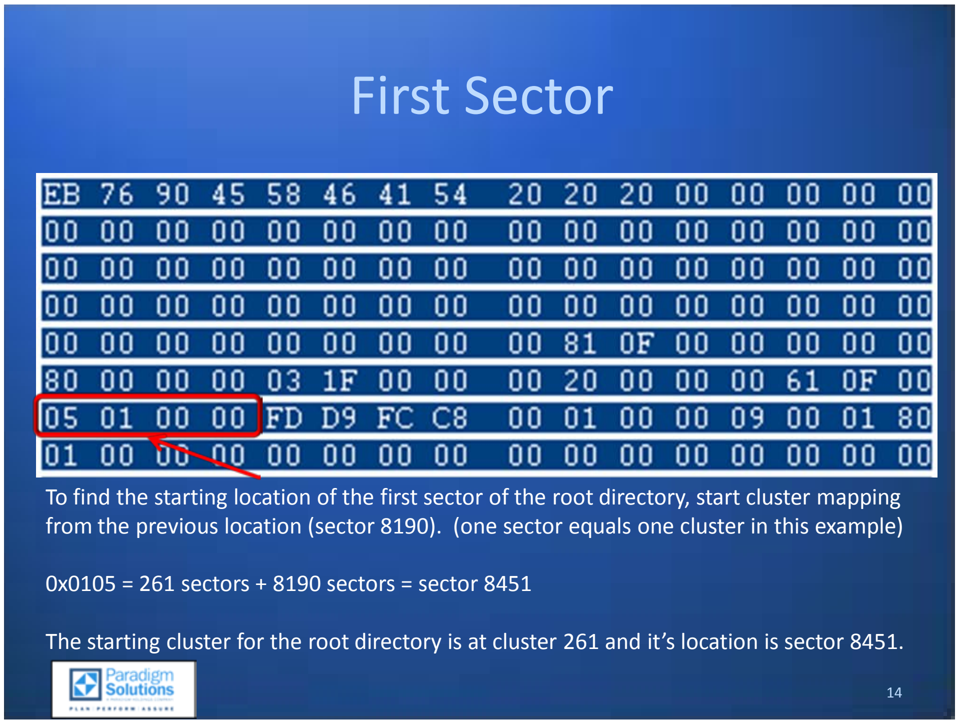
The Bitmap begins on cluster two. To find cluster zero of the file system, subtract (or add*) two clusters to the starting extent of the Bitmap.

0x0F8100 = 8192 – 2 Clusters = Sector 8190

13

Locating the First Logical Cluster

The value of the starting sector for the first data cluster is not readily available by glancing at the volume boot record. The Bitmap begins on cluster two. So to find cluster zero, subtract two clusters from the starting sector of the Bitmap.



First Sector

```
EB 76 90 45 58 46 41 54  20 20 20 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 81 0F 00 00 00 00 00
80 00 00 00 03 1F 00 00  00 20 00 00 00 61 0F 00
05 01 00 00 FD D9 FC C8  00 01 00 00 09 00 01 80
01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

To find the starting location of the first sector of the root directory, start cluster mapping from the previous location (sector 8190). (one sector equals one cluster in this example)

0x0105 = 261 sectors + 8190 sectors = sector 8451

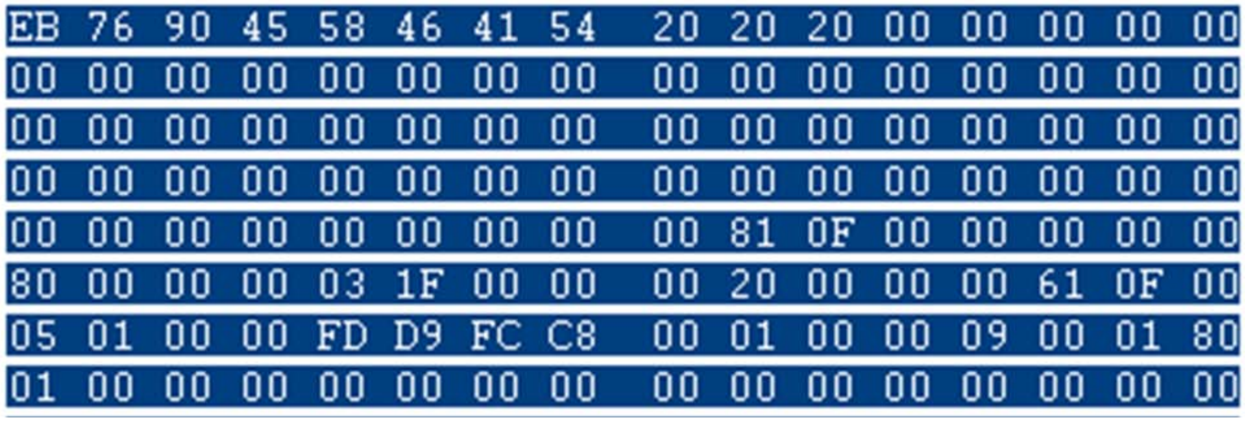
The starting cluster for the root directory is at cluster 261 and it's location is sector 8451.

14

Locating the First Cluster of the Root Directory

To locate the first sector of the root directory, multiply the logical cluster number found in the VBR by the correct number of sectors. As a reference point, the starting location of the bitmap is stored as a physical sector value , and it is located in what can be considered as cluster two. Therefore, the starting sector for cluster zero (and the reference point for all logical addressing) can be determined by subtracting two clusters from the Bitmap's physical location. The cluster offset of the root directory can then be added to this value to find its location on the media.

Example 1:



```
EB 76 90 45 58 46 41 54  20 20 20 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 81 0F 00 00 00 00 00
80 00 00 00 03 1F 00 00  00 20 00 00 00 61 0F 00
05 01 00 00 FD D9 FC C8  00 01 00 00 09 00 01 80
01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

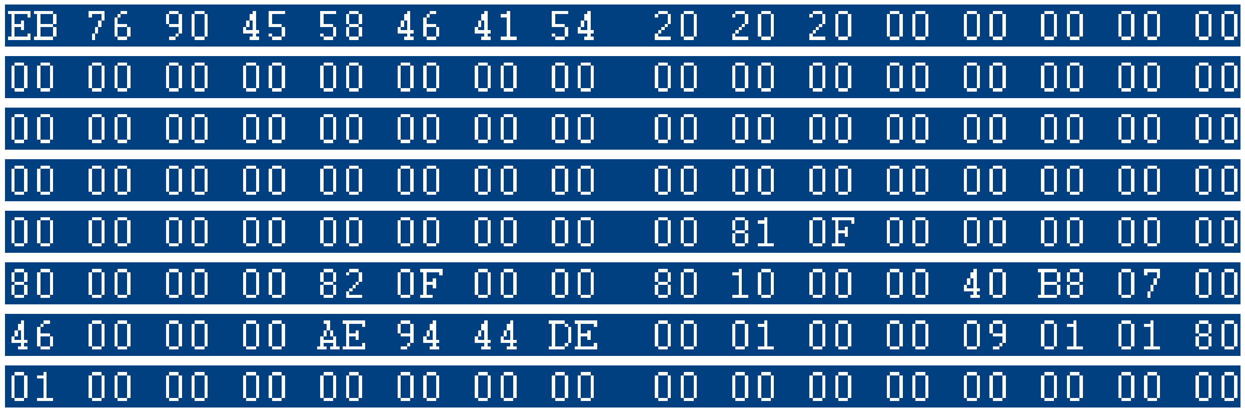
1 Sector = 1 Cluster

The starting sector of the Bitmap is at 8192 (equivalent to cluster two).

Subtracting 2 clusters from 8192 results in 8190.

If the root directory begins on cluster 261, then add to 8190 to find the starting sector to be 8151.

Example 2:



```
EB 76 90 45 58 46 41 54  20 20 20 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 81 0F 00 00 00 00 00
80 00 00 00 82 0F 00 00  80 10 00 00 40 B8 07 00
46 00 00 00 AE 94 44 DE  00 01 00 00 09 01 01 80
01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

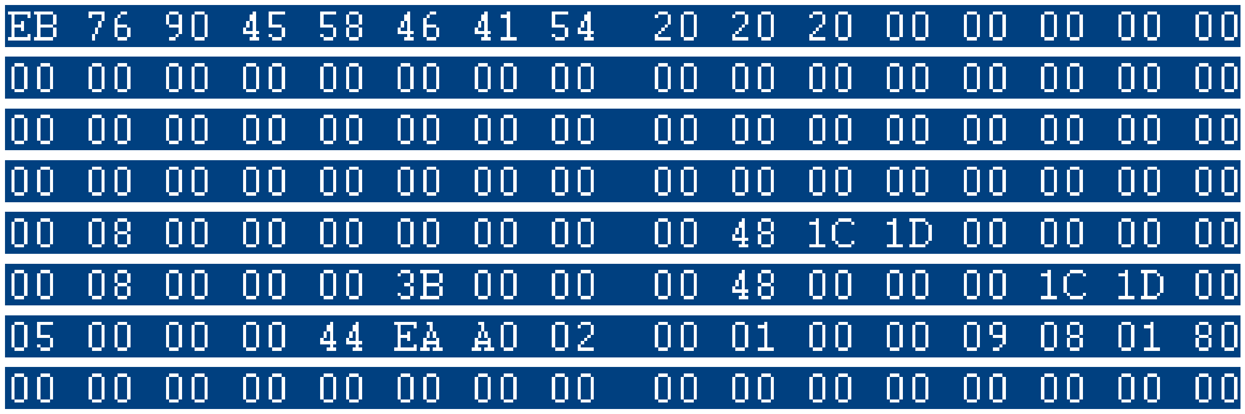
2 Sectors = 1 Cluster

The starting sector of the Bitmap is at 4224 (equivalent to cluster two).

Subtracting 2 clusters from 4224 results in 4220 (2 Clusters * 2 Sectors per Cluster = 4 Sectors, minus 4224).

If the root directory begins on cluster 70, then add 140 (2 Sectors * 70 Clusters) to 4220 to find the starting sector to be 4360.

Example 3:



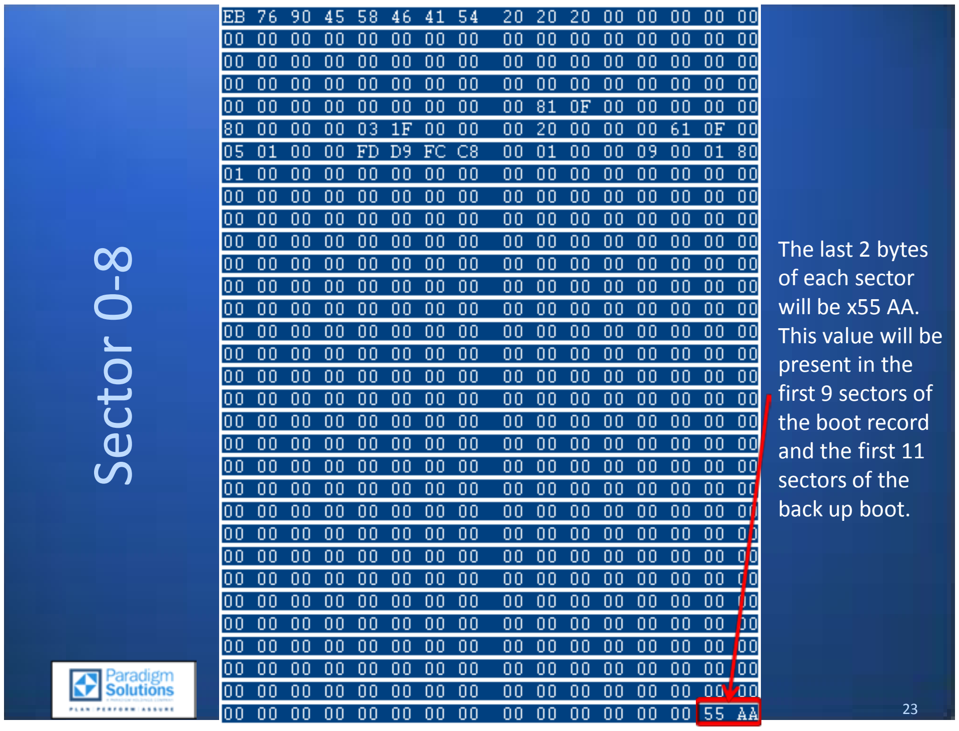
```
EB 76 90 45 58 46 41 54  20 20 20 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 08 00 00 00 00 00 00  00 48 1C 1D 00 00 00 00
00 08 00 00 00 3B 00 00  00 48 00 00 00 1C 1D 00
05 00 00 00 44 EA A0 02  00 01 00 00 09 08 01 80
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

256 Sectors = 1 Cluster

The starting sector of the Bitmap is at 18,432.

Subtracting 2 clusters from 18,432 results in 17,920.

If the root directory begins on cluster 5, then add 1280 (256 Sectors * 5 Clusters) to 17,920 to find the starting sector to be 19,200.



Sector 0-8

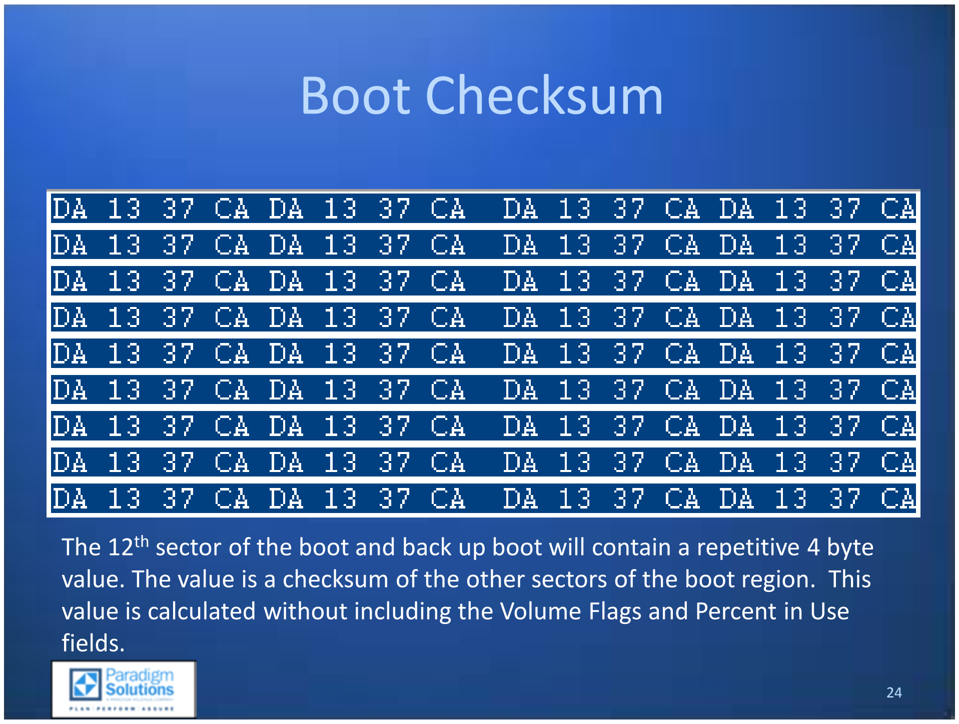
```
EB 76 90 45 58 46 41 54  20 20 20 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 81 0F 00 00 00 00 00
80 00 00 00 03 1F 00 00  00 20 00 00 00 61 0F 00
05 01 00 00 FD D9 FC C8  00 01 00 00 09 00 01 80
01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 AA
```

The last 2 bytes of each sector will be x55 AA. This value will be present in the first 9 sectors of the boot record and the first 11 sectors of the back up boot.

23

Boot Signature – 0x55 AA

The signature 0x55 AA will be present as the last two bytes of the Volume Boot Record. This signature will be located at the end of each of the first 9 sectors of the volume. Sector 11 contains data as listed on the next page. Sector 12-20 will also contain the boot signature as the VBR is backed up in its entirety. This signature can cause false hits when using a script or regular expression to search for boot records.



Boot Checksum

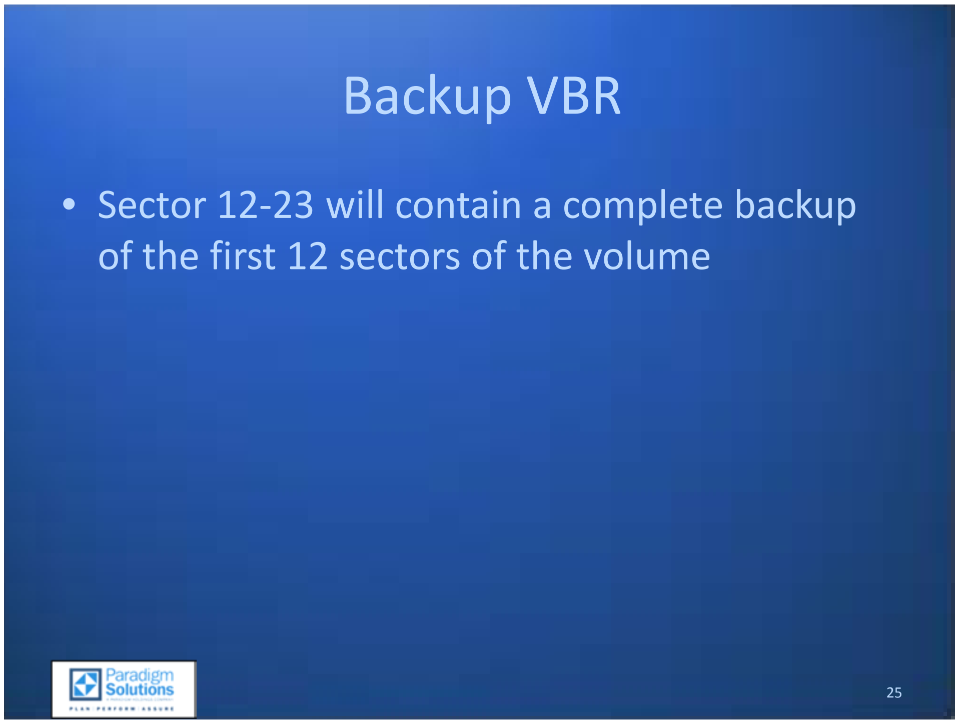
```
DA 13 37 CA DA 13 37 CA  DA 13 37 CA DA 13 37 CA
DA 13 37 CA DA 13 37 CA  DA 13 37 CA DA 13 37 CA
DA 13 37 CA DA 13 37 CA  DA 13 37 CA DA 13 37 CA
DA 13 37 CA DA 13 37 CA  DA 13 37 CA DA 13 37 CA
DA 13 37 CA DA 13 37 CA  DA 13 37 CA DA 13 37 CA
DA 13 37 CA DA 13 37 CA  DA 13 37 CA DA 13 37 CA
DA 13 37 CA DA 13 37 CA  DA 13 37 CA DA 13 37 CA
DA 13 37 CA DA 13 37 CA  DA 13 37 CA DA 13 37 CA
DA 13 37 CA DA 13 37 CA  DA 13 37 CA DA 13 37 CA
```

The 12th sector of the boot and back up boot will contain a repetitive 4 byte value. The value is a checksum of the other sectors of the boot region. This value is calculated without including the Volume Flags and Percent in Use fields.

24

Boot Checksum

Sector 11 contains a repetitive 32 bit value. This value, generated during format and is copied in the backup VBR, is a checksum of the other areas of the boot region, not including the volume flags or the percent-in-use fields.



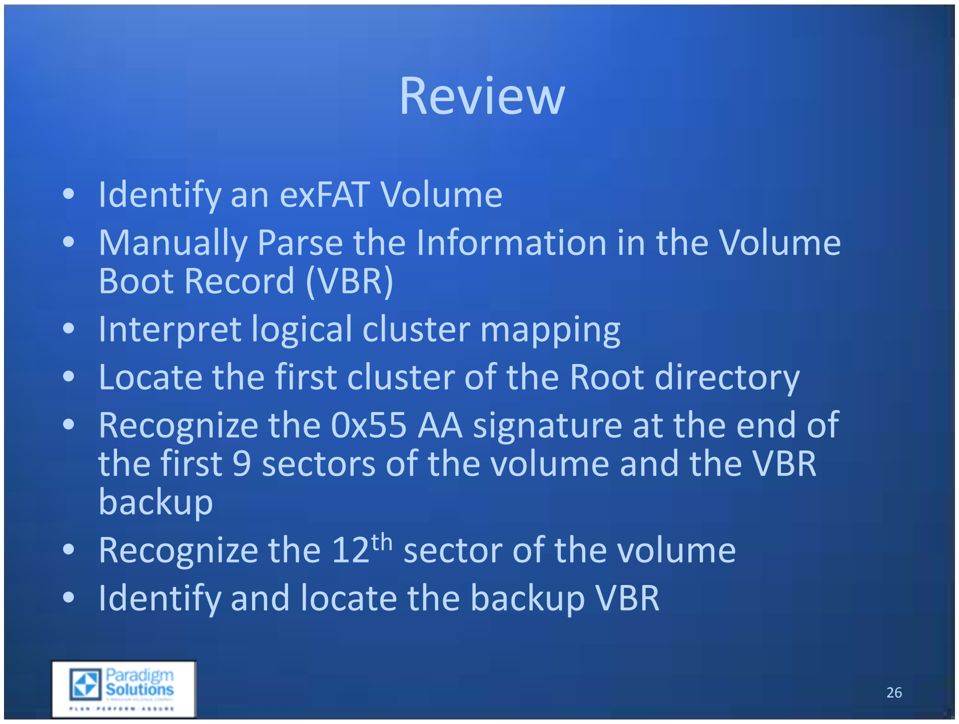
Backup VBR

- Sector 12-23 will contain a complete backup of the first 12 sectors of the volume

25

Backup VBR

The first 12 sectors (sector 0-11) will be stored in a backup in sector 12-23. These two VBRs will normally be identical, which can be verified using a hash sum.



Review

- Identify an exFAT Volume
- Manually Parse the Information in the Volume Boot Record (VBR)
- Interpret logical cluster mapping
- Locate the first cluster of the Root directory
- Recognize the 0x55 AA signature at the end of the first 9 sectors of the volume and the VBR backup
- Recognize the 12th sector of the volume
- Identify and locate the backup VBR

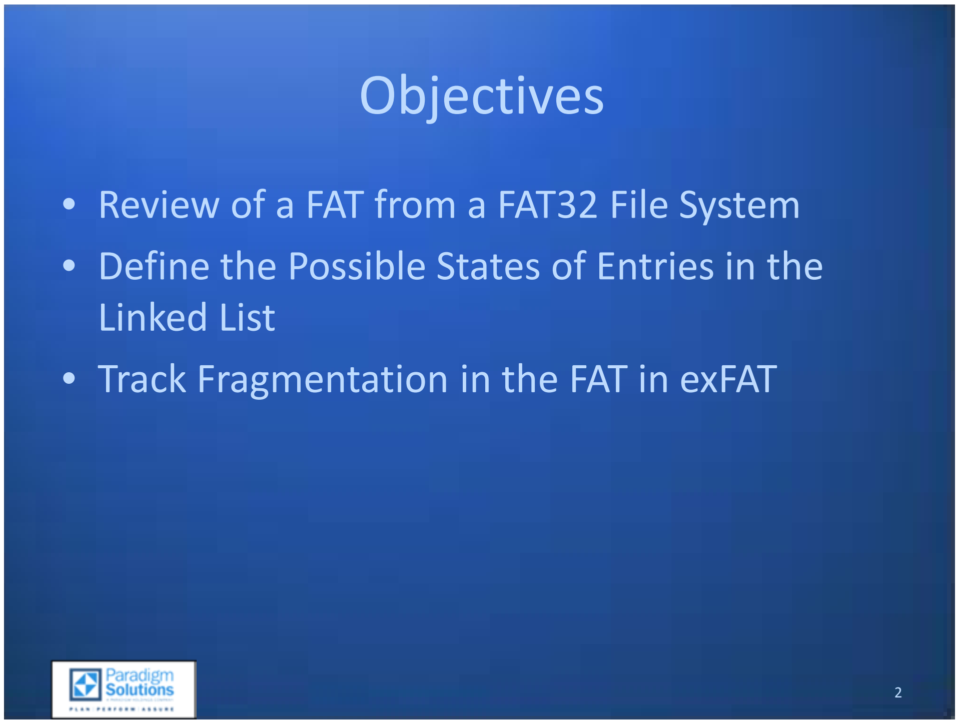
26

Review

With the completion of this section, review how to do the following:

- Identify an exFAT Volume
- Manually Parse the Information in the Volume Boot Record (VBR)
- Interpret logical cluster mapping
- Locate the first cluster of the Root directory
- Recognize the 0x55 AA signature at the end of the first 9 sectors of the volume and the VBR backup
- Recognize the 12th sector of the volume
- Identify and locate the backup VBR

Linked List



Objectives

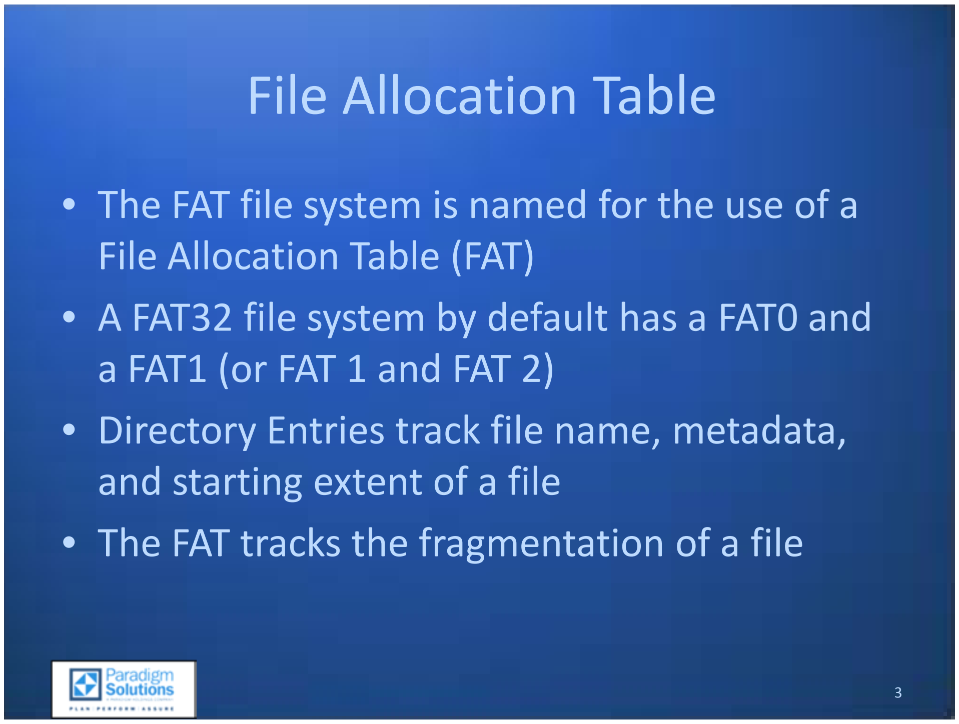
- Review of a FAT from a FAT32 File System
- Define the Possible States of Entries in the Linked List
- Track Fragmentation in the FAT in exFAT

2

Objectives

The examiner will be able to understand and explain the following after the completion of this section:

- The FAT (File Allocation Table) from a FAT32 File System
- The Possible States of Entries in the Linked List
- The means to track Fragmentation in the FAT in exFAT



File Allocation Table

- The FAT file system is named for the use of a File Allocation Table (FAT)
- A FAT32 file system by default has a FAT0 and a FAT1 (or FAT 1 and FAT 2)
- Directory Entries track file name, metadata, and starting extent of a file
- The FAT tracks the fragmentation of a file

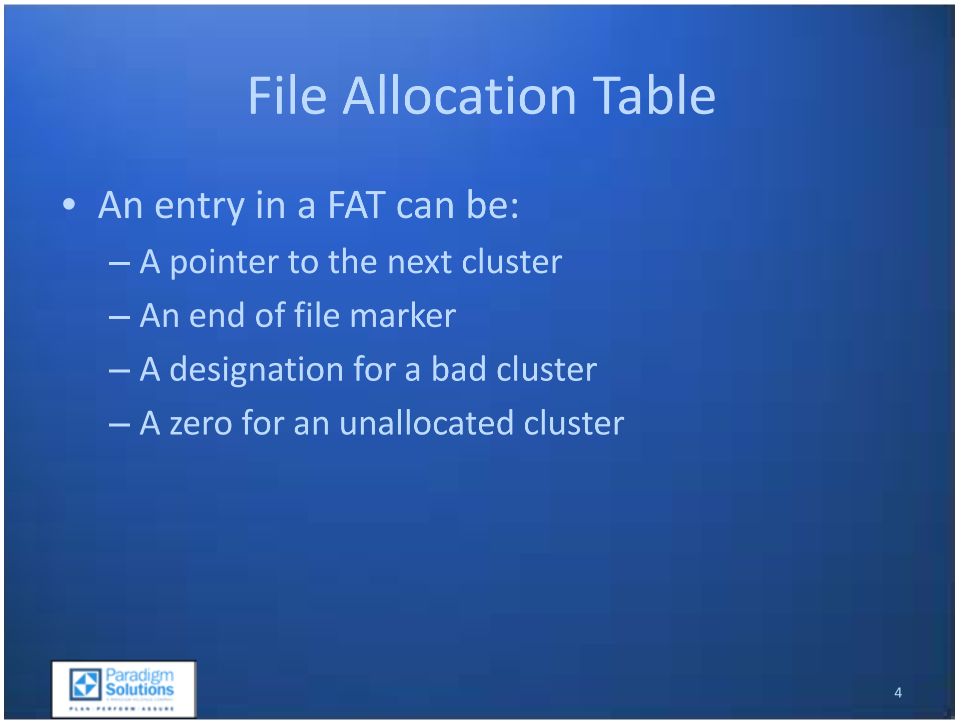
3

File Allocation Table

The FAT file system is used frequently in training classes because of the vast familiarity with this relatively simple file system. A FAT file system has a VBR, FAT and backup FAT, and Root Directory. With FAT32, only the first cluster of the Root Directory was static and could be considered to have been in the "System" area of a volume.

In a FAT file system, the FAT itself tracked cluster allocation, which was defined in the VBR, as well as the cluster fragmentation. FAT used the directory entry to define the size of a file and the starting extent, along with whether the file was deleted or allocated.

Transactional FAT options (TFAT) write changes to one FAT without committing those changes to the other FAT until the write operation has completed successfully. As a file system, FAT is quite simple, not requiring the overhead of NTFS.



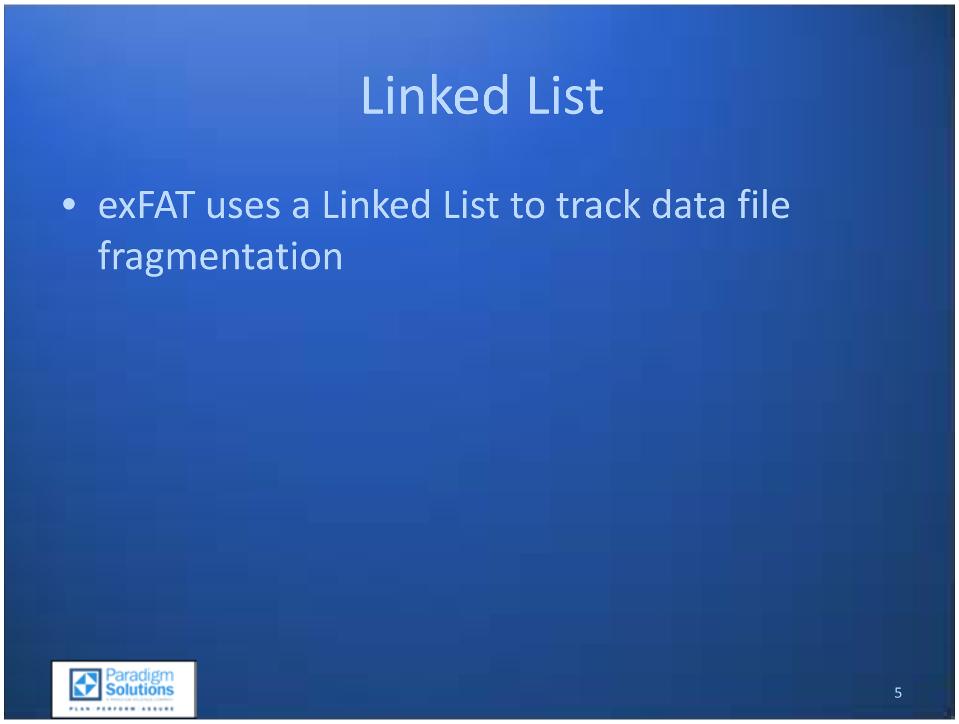
File Allocation Table

- An entry in a FAT can be:
  - A pointer to the next cluster
  - An end of file marker
  - A designation for a bad cluster
  - A zero for an unallocated cluster

4

File Allocation Table Entries

With the FAT file system, a FAT entry could be a pointer to the next cluster, an end of file marker, a designation for a bad cluster, or an unallocated cluster.



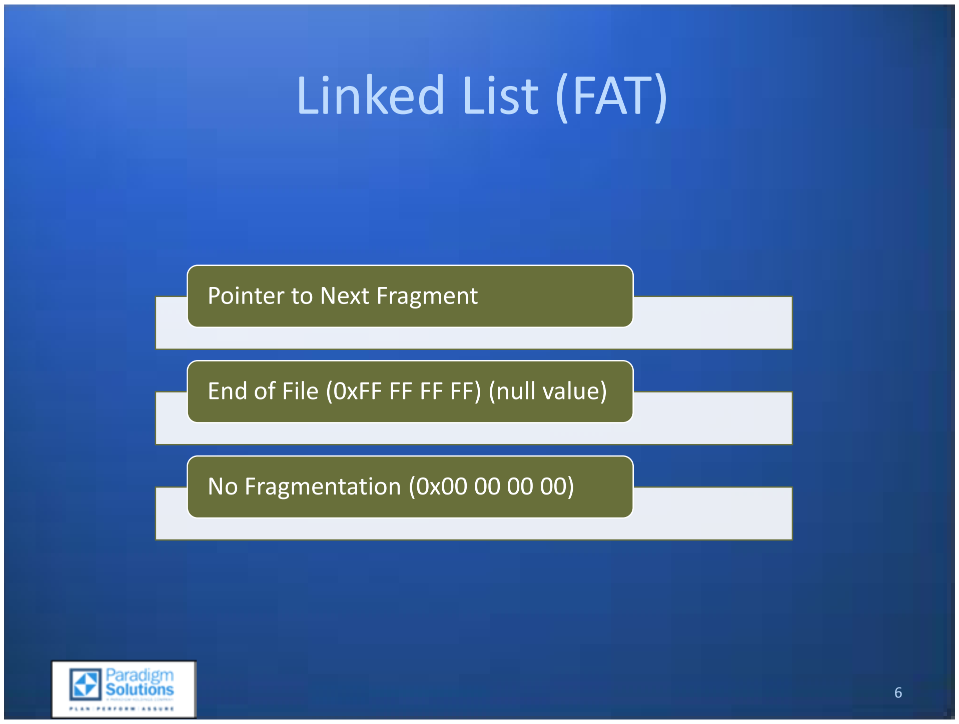
Linked List

- exFAT uses a Linked List to track data file fragmentation

5

Linked List

A linked list contains and manages pointers and data sets. Dynamic linked lists can point forward and backward. Testing shows that the exFAT file system uses a simple linked list to track fragmentation. The list is nothing more than a pointer to the next cluster if a file is fragmented. (Parlante, 2001)



Linked List (FAT)

- Pointer to Next Fragment
- End of File (0xFF FF FF FF) (null value)
- No Fragmentation (0x00 00 00 00)

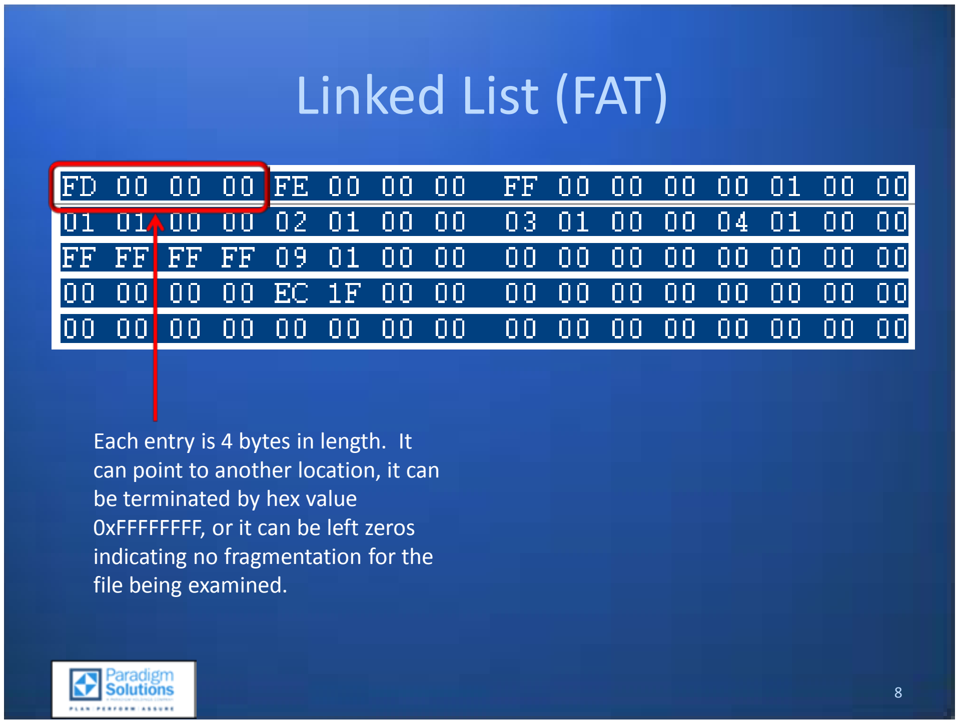
6

FAT Entries

With exFAT, the FAT can be a series of values. The linked list can point to the next fragment (cluster) in a chain, signify the end of a file's fragmentation by storing a null value of 0xFFFFFFFF, or store the value of 0x00000000 to indicate there is no chain or fragmentation within the referenced clusters. The value 0x00000000 seems to be used for both unallocated areas and for allocated areas (again, signifying no fragmentation). It is likely that the bitmap checks for allocation status, and the linked list only stores values if a file has to be stored in a fragmented manner. Critical structures, such as the Root Directory and the Bitmap, appear to always have clusters mapped in the linked list and are never recorded as 0x00000000.

exFAT includes Special Designators. The first entry in the FAT is the media descriptor field and it must have the most significant byte set to 0xF8 (0xFFFFFFF8). A bad cluster will have the most significant byte set to of 0xF7 (0xFFFFFFF7).

Like FAT12/16/32, exFAT cannot point backwards in the FAT chain. Each link must be a higher cluster on the volume.



Linked List (FAT)

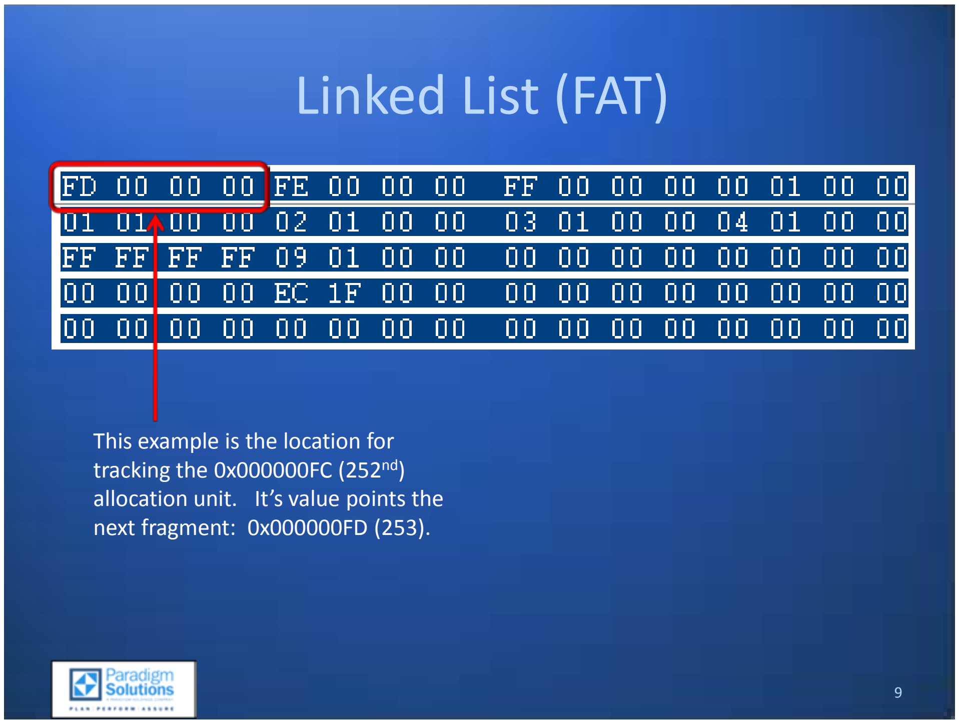
```
FD 00 00 00 FE 00 00 00  FF 00 00 00 00 01 00 00
01 01 00 00 02 01 00 00  03 01 00 00 04 01 00 00
FF FF FF FF 09 01 00 00  00 00 00 00 00 00 00 00
00 00 00 00 EC 1F 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

Each entry is 4 bytes in length. It can point to another location, it can be terminated by hex value 0xFFFFFFFF, or it can be left zeros indicating no fragmentation for the file being examined.

8

Linked List Pointers

Currently, no automated software packages can trace the exFAT linked list such as Disk Edit does for FAT. To follow the list, an examiner must calculate the math manually. The pointers are stored in 32 bit (4 byte) values and can be measured by offset from the start of the Linked List as determined from the VBR. The value stored in the record will either point to the next cluster or will terminate if a file is either fragmented or is a critical system file. As noted above, the value of zeros appears to be stored in the list not only if the cluster is unallocated but also if the file stored in the cluster is not fragmented.



Linked List (FAT)

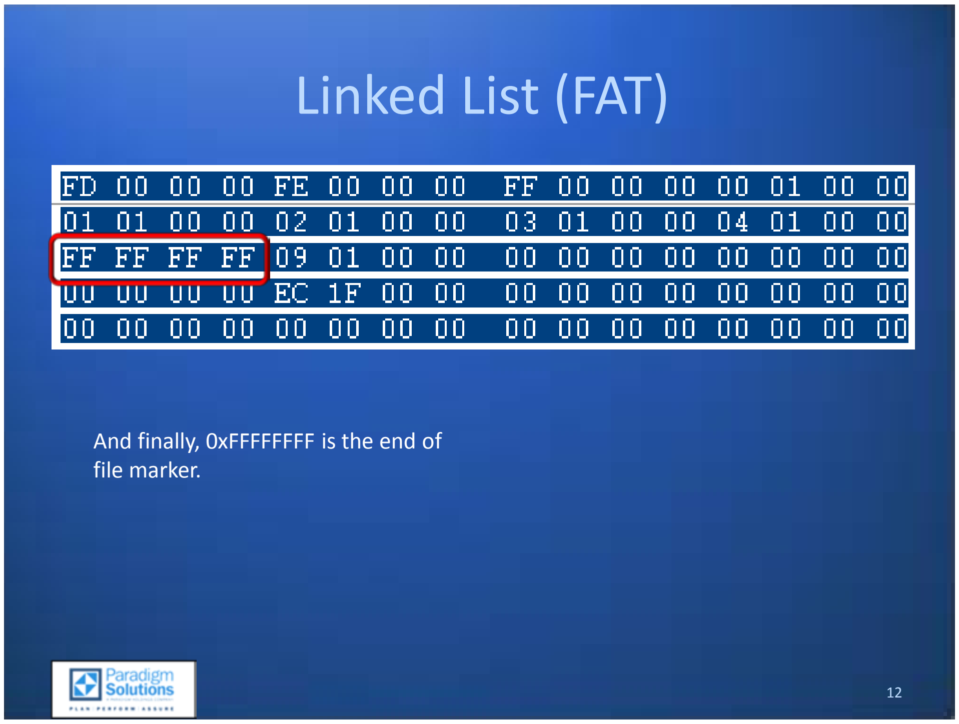
```
FD 00 00 00 FE 00 00 00  FF 00 00 00 00 01 00 00
01 01 00 00 02 01 00 00  03 01 00 00 04 01 00 00
FF FF FF FF 09 01 00 00  00 00 00 00 00 00 00 00
00 00 00 00 EC 1F 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

This example is the location for tracking the 0x000000FC (252nd) allocation unit. It's value points the next fragment: 0x000000FD (253).

9

Linked List Pointers

To exemplify based on the screen capture, each record (or pointer) is 4 bytes. The record stored in 0xFC (252) points to 0xFD (253) which then points to 0xFE (254) and so on until the terminated value (0xFFFFFFFF) is reached.



Linked List (FAT)

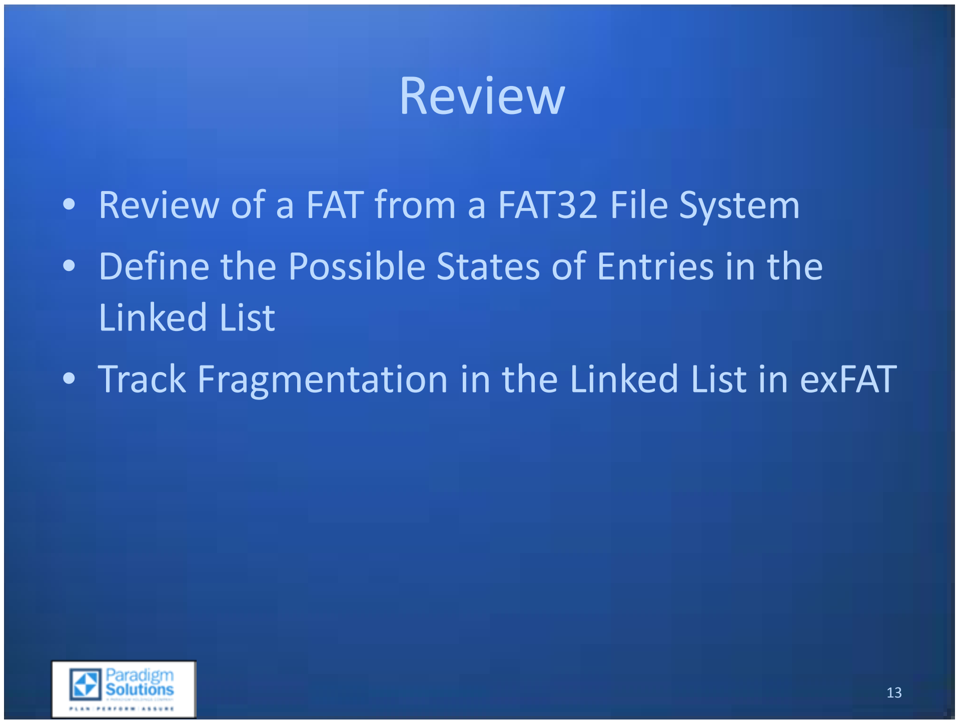
```
FD 00 00 00 FE 00 00 00  FF 00 00 00 00 01 00 00
01 01 00 00 02 01 00 00  03 01 00 00 04 01 00 00
FF FF FF FF 09 01 00 00  00 00 00 00 00 00 00 00
00 00 00 00 EC 1F 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

And finally, 0xFFFFFFFF is the end of file marker.

12

Linked List Null Value

The end of the list is marked with 0xFFFFFFFF as previously noted.



Review

- Review of a FAT from a FAT32 File System
- Define the Possible States of Entries in the Linked List
- Track Fragmentation in the Linked List in exFAT

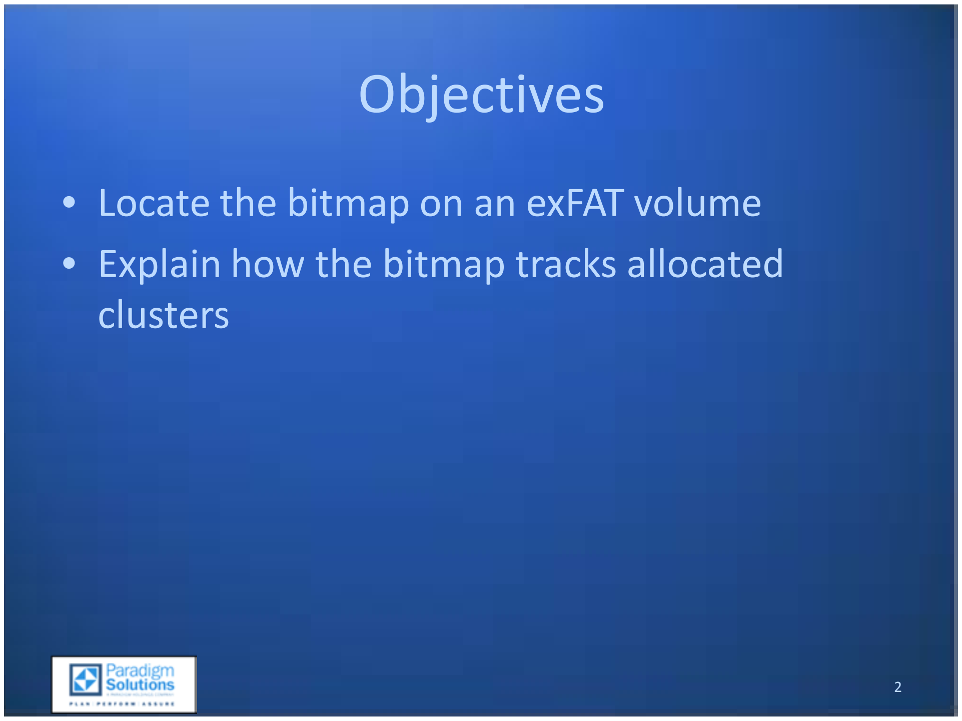
13

Review

Having completed this section, the examiner will be able to do the following:

- Review a FAT (File Allocation Table) from a FAT32 File System
- Define the Possible States of Entries in the Linked List
- Track Fragmentation in the Linked List in exFAT

Bitmap



Objectives

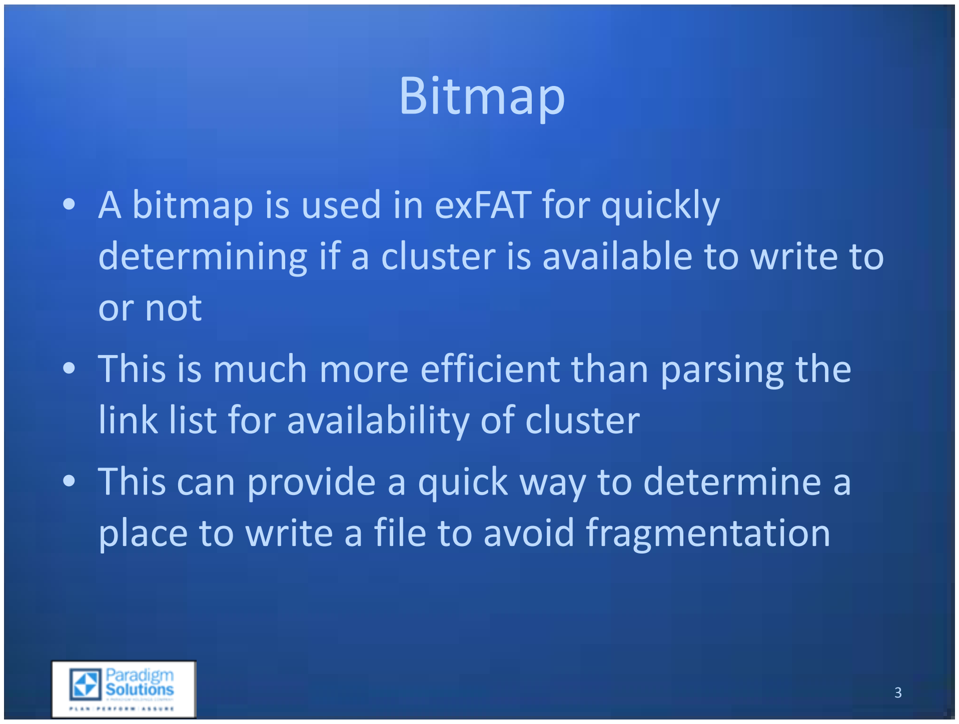
- Locate the bitmap on an exFAT volume
- Explain how the bitmap tracks allocated clusters

2

Objectives

After completing this section, the analyst will have the ability to:

- Locate the Bitmap on an exFAT Volume
- Explain How the Bitmap Tracks Allocated Clusters



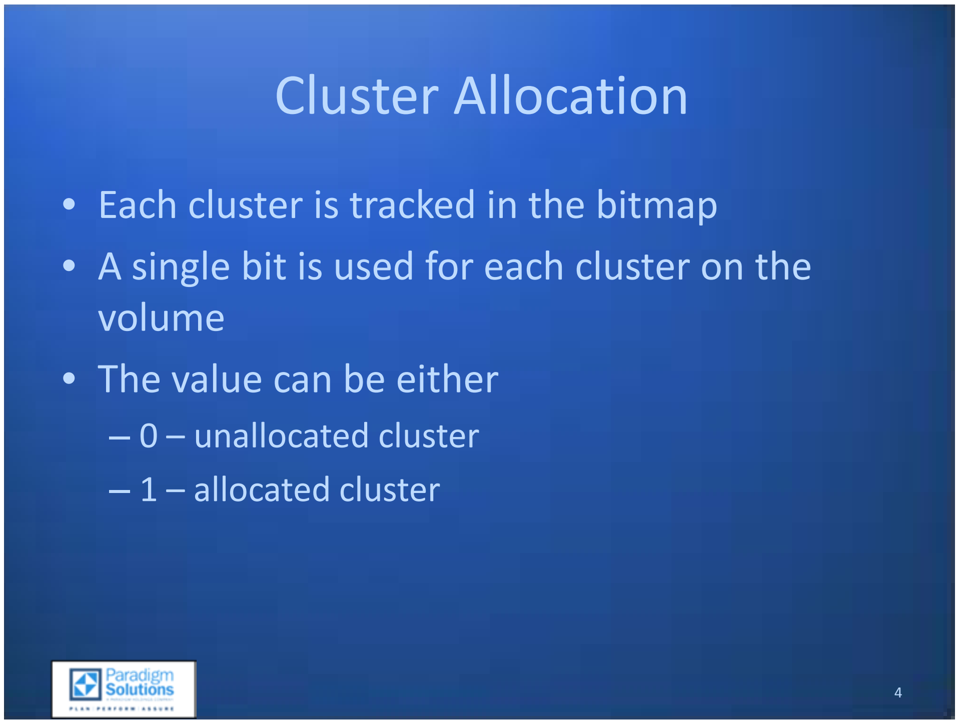
Bitmap

- A bitmap is used in exFAT for quickly determining if a cluster is available to write to or not
- This is much more efficient than parsing the link list for availability of cluster
- This can provide a quick way to determine a place to write a file to avoid fragmentation

3

Bitmap

A bitmap as used in a file system is a consecutive array of bits. Each bit is spatially relative to each cluster on the file system. The file system can quickly reference the bitmap to determine which series of clusters is available to store a file, as the file system can efficiently scan its relatively small size for blocks of unallocated space, with an emphasis on locating space large enough to avoid fragmentation. This is far more efficient than having to load and parse out a FAT or linked list, which in exFAT is 32 bits per cluster.



Cluster Allocation

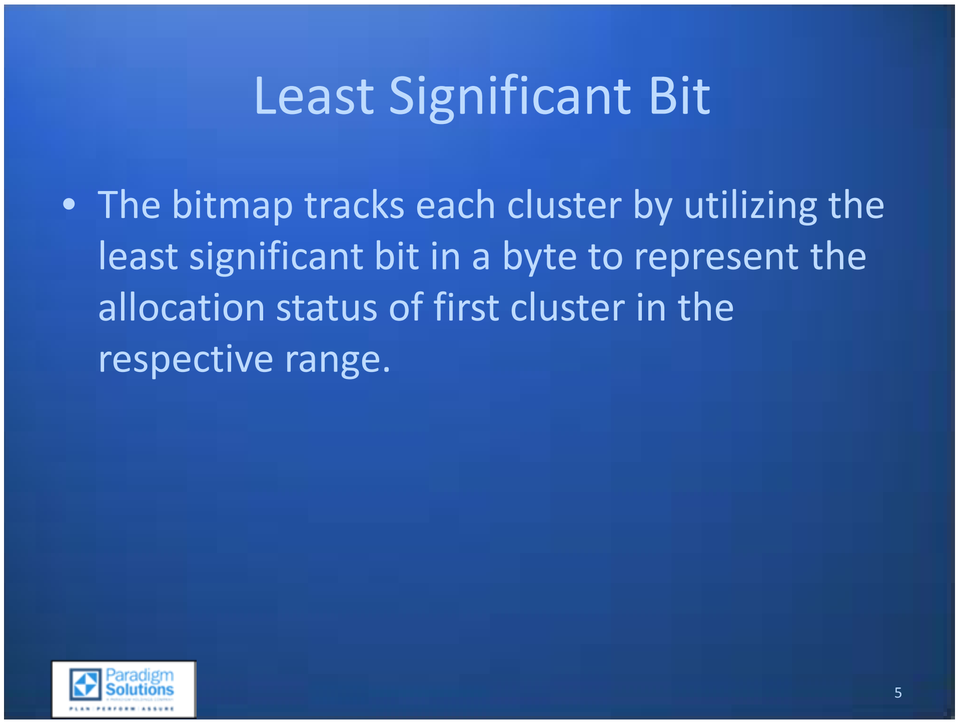
- Each cluster is tracked in the bitmap
- A single bit is used for each cluster on the volume
- The value can be either
  - 0 – unallocated cluster
  - 1 – allocated cluster

4

Cluster Allocation

Cluster allocation is tracked in the bitmap. Each cluster is represented by a single bit in the bitmap for the entire volume. The values are simply zero (0), meaning an unallocated cluster that's free for use, or one (1), an allocated cluster which contains file or folder data.

Importantly, the first bit of the bitmap points directly to cluster two, the first addressable cluster on the volume.



Least Significant Bit

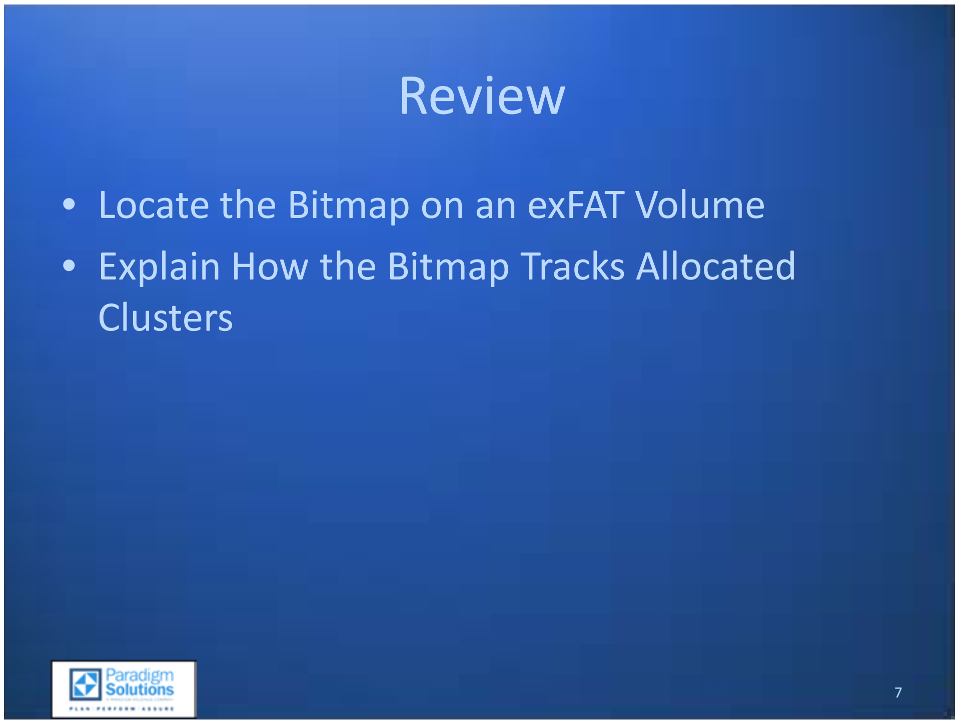
- The bitmap tracks each cluster by utilizing the least significant bit in a byte to represent the allocation status of first cluster in the respective range.

5

Least Significant Bit

The least significant bit in a byte tracks the clusters. If allocated, the very first cluster would be represented by a hex value of 0x01 or binary value of 0000|0001. If the first and eighth cluster were used, it could be represented by a value of 0x81 or 1000|0001.

To find the allocation status of a specific cluster, navigate to the bitmap as listed in the VBR using a hex editor. Next, locate the offset in the bitmap by dividing the value by eight to represent the bits displayed as bytes. Fractions will indicate the bit position in the byte.



Review

- Locate the Bitmap on an exFAT Volume
- Explain How the Bitmap Tracks Allocated Clusters

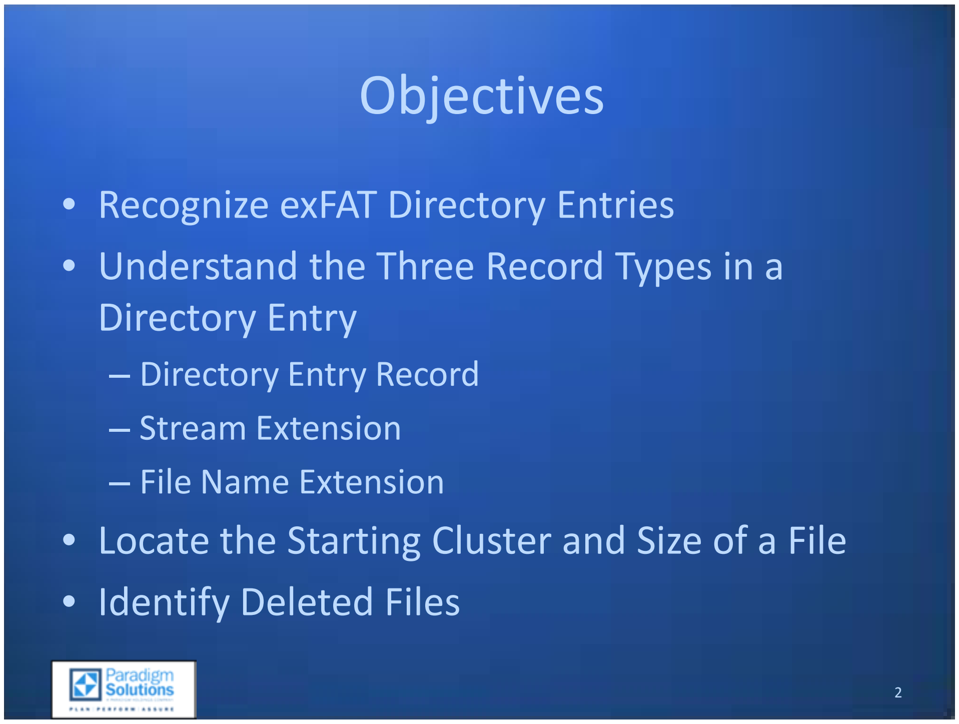
7

Review

After completing this section, the analyst is able to:

- Locate the Bitmap on an exFAT Volume
- Explain How the Bitmap Tracks Allocated Clusters

Directory Entries



Objectives

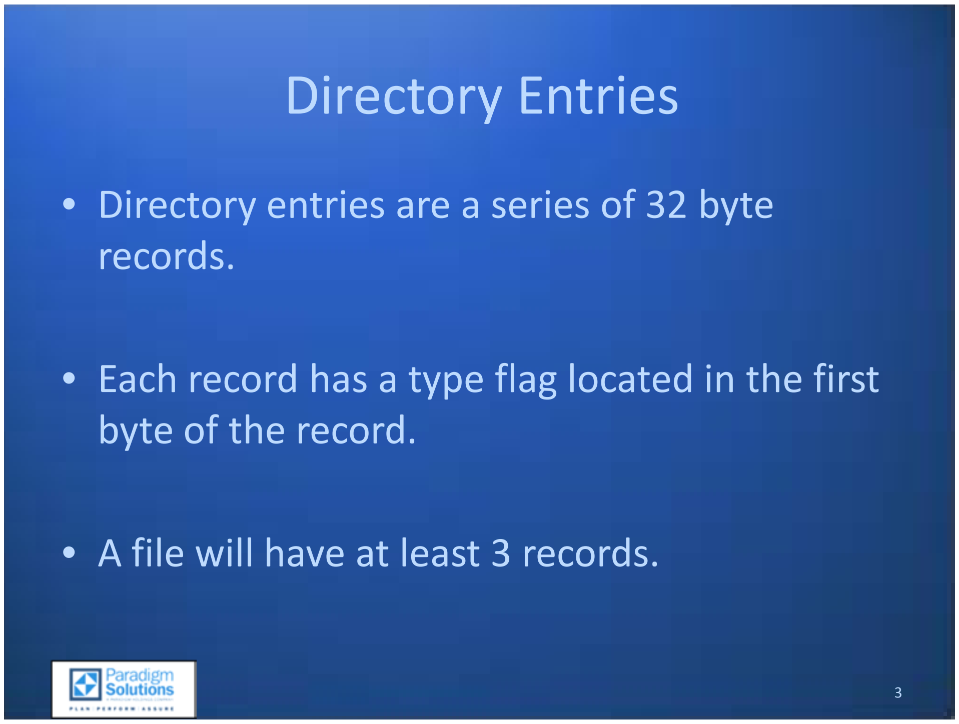
- Recognize exFAT Directory Entries
- Understand the Three Record Types in a Directory Entry
  - Directory Entry Record
  - Stream Extension
  - File Name Extension
- Locate the Starting Cluster and Size of a File
- Identify Deleted Files

2

Objectives

Recognizing the need to address the new exFAT file system, this module will provide the examiner with the ability to:

- Recognize exFAT Directory Entries
- Understand the Three Record Types in a Directory Entry
  - Directory Entry Record
  - Stream Extension
  - File Name Extension
- Locate the Starting Cluster and Size of a File
- Identify Deleted Files



Directory Entries

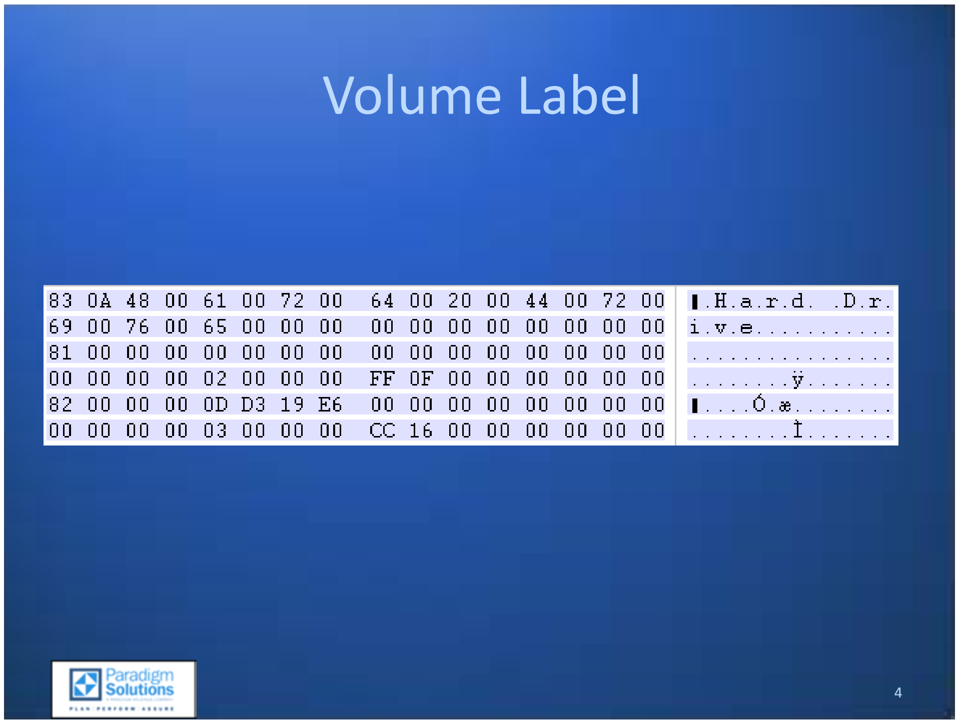
- Directory entries are a series of 32 byte records.
- Each record has a type flag located in the first byte of the record.
- A file will have at least 3 records.

3

Directory Entries

Directory entries in exFAT store, in addition to the file name, metadata about the file such as the starting location, size of the file, and DOS type attributes (hidden, system, archive).

At least three records will exist for any file or folder on the volume. These records are stored in 32 bytes, with the first byte of the record identifying the record type and, therefore, the data it is storing.



Volume Label

```
83 0A 48 00 61 00 72 00  64 00 20 00 44 00 72 00  ▌.H.a.r.d. .D.r.
69 00 76 00 65 00 00 00  00 00 00 00 00 00 00 00  i.v.e...........
81 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00 00 00 00 02 00 00 00  FF 0F 00 00 00 00 00 00  ........ÿ.......
82 00 00 00 0D D3 19 E6  00 00 00 00 00 00 00 00  ▌....Ó.æ........
00 00 00 00 03 00 00 00  CC 16 00 00 00 00 00 00  ........Ì.......
```

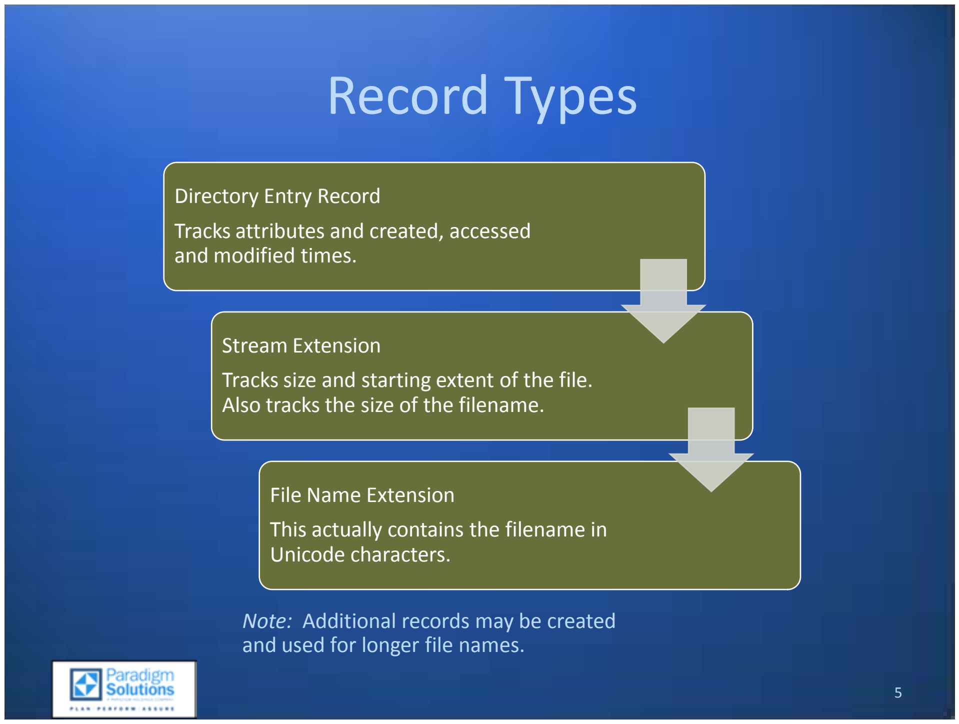
4

Volume Label

The volume label will be the first directory stored in the root directory. The volume label will use three records similar to a file or folder entry. The identifier of this series of records will be 0x83, 0x81, and 0x82. The volume name will be stored in the first record (identifier 0x83). The second byte of the record will be the number of Unicode characters used for the volume name. The third and fourth bytes of the record will be the first Unicode character of the volume name.

Records of type 0x81 refer to the size and location of the bitmap. Records of the type 0x82 refer to the location and size of the Up-Case table.

If the volume was not given a label, the first value for the first entry will be 0x03 rather than 0x83.



Record Types

Directory Entry Record

Tracks attributes and created, accessed and modified times.

Stream Extension

Tracks size and starting extent of the file. Also tracks the size of the filename.

File Name Extension

This actually contains the filename in Unicode characters.

Note: Additional records may be created and used for longer file names.

5

Record Types

Directory entries will each have a minimum of three records. One directory entry record tracks DOS-type attributes and time stamps for the file. A stream extension record tracks the starting extent of a file, size of the file, and the length of the file name in Unicode characters. The final record in a directory entry will be the actual file name or the file name extension.

The record types and their identifier are listed in Table 3 – exFAT Directory Entry Record Type Values.

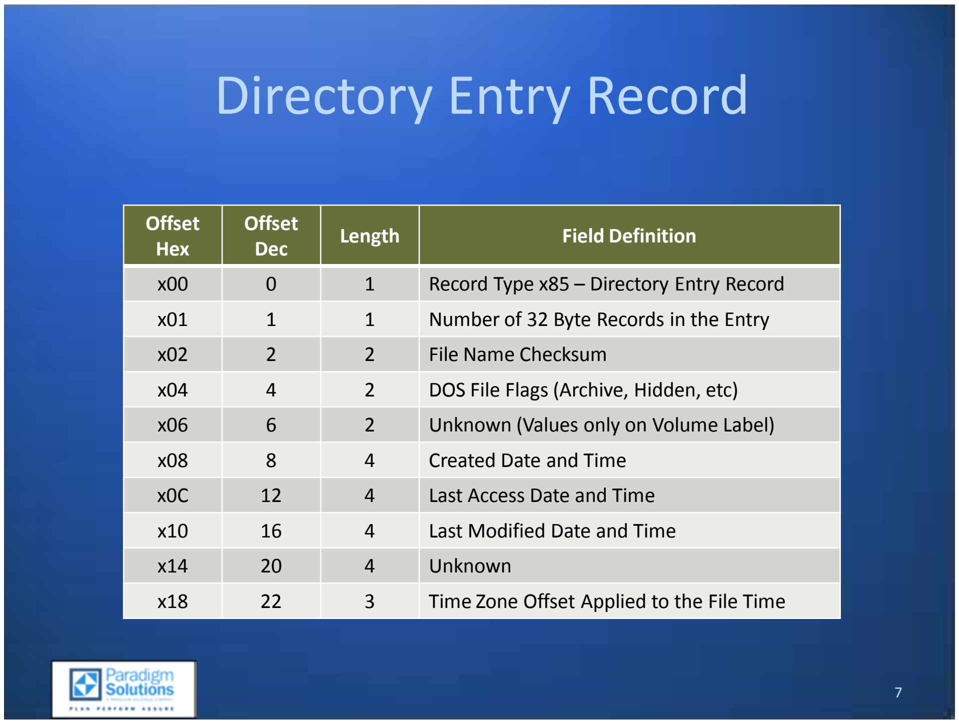
Offset Hex	Offset Dec	Length	Field Definition
x85	0	1	Directory Entry Record
x83	1	1	Volume Name Record
x82	2	2	Up-Case Table Logical Location and Size
x81	4	2	Bitmap Logical Location and Size
xC0	6	2	Directory Entry Record, Stream Extension
xC1	8	4	Directory Entry Record, File Name Extension

Table 3 – exFAT Directory Entry Record Type Values

Volume Name and the Up-Case table will only be present in the root directory. The volume name will maintain the volume label as given by the user. As previously mentioned, if no label was given, this entry will be 0x03 rather than 0x83.

The Up-Case table is used by the file system to maintain a case-insensitive, case-preserving file system. This entry will have the logical cluster location of the Up-Case table at offset 0x14 (20d) and the size in bytes at offset 0x18 (24).

The bitmap logical location and size will have an entry here also. The logical cluster location can be found at offset 0x14 (20d) and the size in bytes at offset 0x18 (24).



Directory Entry Record

Offset Hex	Offset Dec	Length	Field Definition
x00	0	1	Record Type x85 – Directory Entry Record
x01	1	1	Number of 32 Byte Records in the Entry
x02	2	2	File Name Checksum
x04	4	2	DOS File Flags (Archive, Hidden, etc)
x06	6	2	Unknown (Values only on Volume Label)
x08	8	4	Created Date and Time
x0C	12	4	Last Access Date and Time
x10	16	4	Last Modified Date and Time
x14	20	4	Unknown
x18	22	3	Time Zone Offset Applied to the File Time

7

Directory Entry Record

The Directory Entry Record – record type 0x85 – keeps track of information about the file. It includes a count of all other records associated with a directory entry, DOS file flags or attributes, and the MAC information. There are two fields whose purpose has not been identified at the time of this writing. Each value referenced in Table 8 - exFAT Master Record Data is an offset relative to the Directory Entry Record. Dates and times are recorded in DOS 32 bit format.

The time zone offset is sometimes stored in the master record. Its presence is occurs in Vista SP2, Windows 7, and XP SP3. When it is stored, each individual byte applies to the created, accessed, and modified times, respectively. For example, on an XP machine with the exFAT drivers installed and set to GMT-5:00 with daylight time active, the three bytes were 0xF0F0F0. That equates to 240 minutes, or 4 hours, applied to each of the times (created is GMT-4, accessed is GMT-4, and modified is GMT-4).

The time zone offset is stored as 15 minute intervals in a 7 bit signed integer (complement of twos form). The high order bit is ignored for purposes of calculating the time zone offset.

Example One EDT:

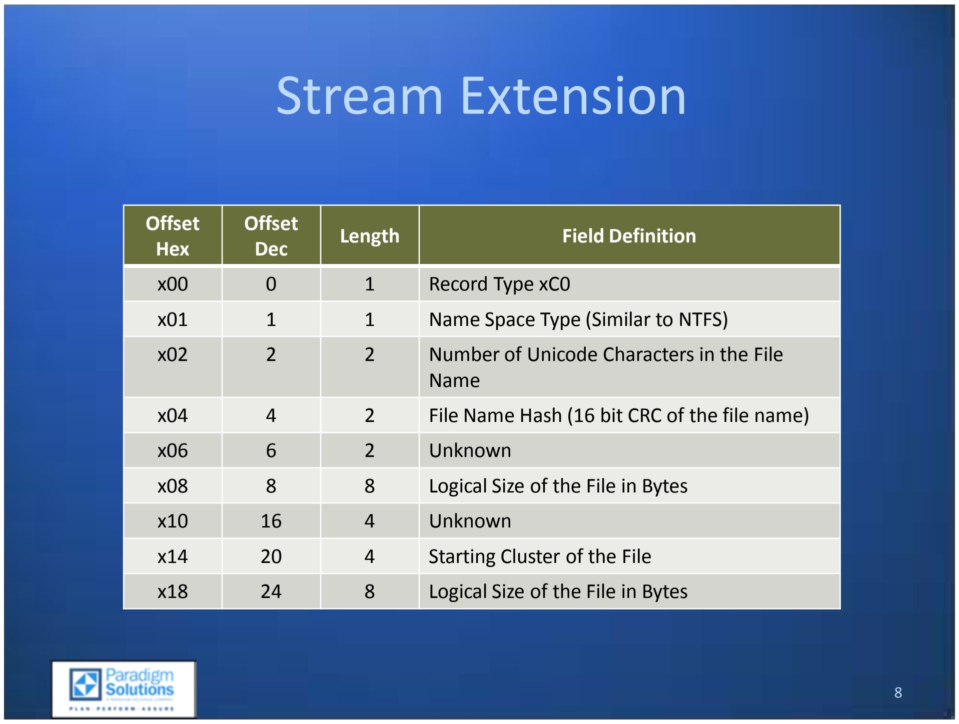
- The stored time zone offset above is 0xF0
- 0xF0 = 1111|0000
- Dropping the most significant bit results in 111|0000
- Sign the value (-64 + 32 +16 = -16)
- Multiply the value by 15 minute intervals (-16 * 15 = -240)
- Therefore when the time stamp was applied, the local machine was set to UTC -4 (EDT)

Example Two UTC:

- The stored time zone offset is 0x80
- 0x80 = 1000|0000
- Dropping the most significant bit results in 000|0000
- Sign the value (-0 + 0 = 0)
- Multiply the value by 15 minute intervals (0 * 15 = 0)
- Therefore when the time stamp was applied the local machine was set to UTC

Offset Hex	Offset Dec	Length	Field Definition
x00	0	1	Record Type x85 – Directory Entry Record
x01	1	1	Number of 32 Byte Records in the Entry
x02	2	2	File Name Checksum
x04	4	2	DOS File Flags (Archive, Hidden, etc)
x06	6	2	Unknown (Values only on Volume Label)
x08	8	4	Created Date and Time
x0C	12	4	Last Access Date and Time
x10	16	4	Last Modified Date and Time
x14	20	4	Unknown
x18	22	3	Time Zone Offset Applied to the File Time

Table 4 - exFAT Directory Entry Record Data



Stream Extension

Offset Hex	Offset Dec	Length	Field Definition
x00	0	1	Record Type xC0
x01	1	1	Name Space Type (Similar to NTFS)
x02	2	2	Number of Unicode Characters in the File Name
x04	4	2	File Name Hash (16 bit CRC of the file name)
x06	6	2	Unknown
x08	8	8	Logical Size of the File in Bytes
x10	16	4	Unknown
x14	20	4	Starting Cluster of the File
x18	24	8	Logical Size of the File in Bytes

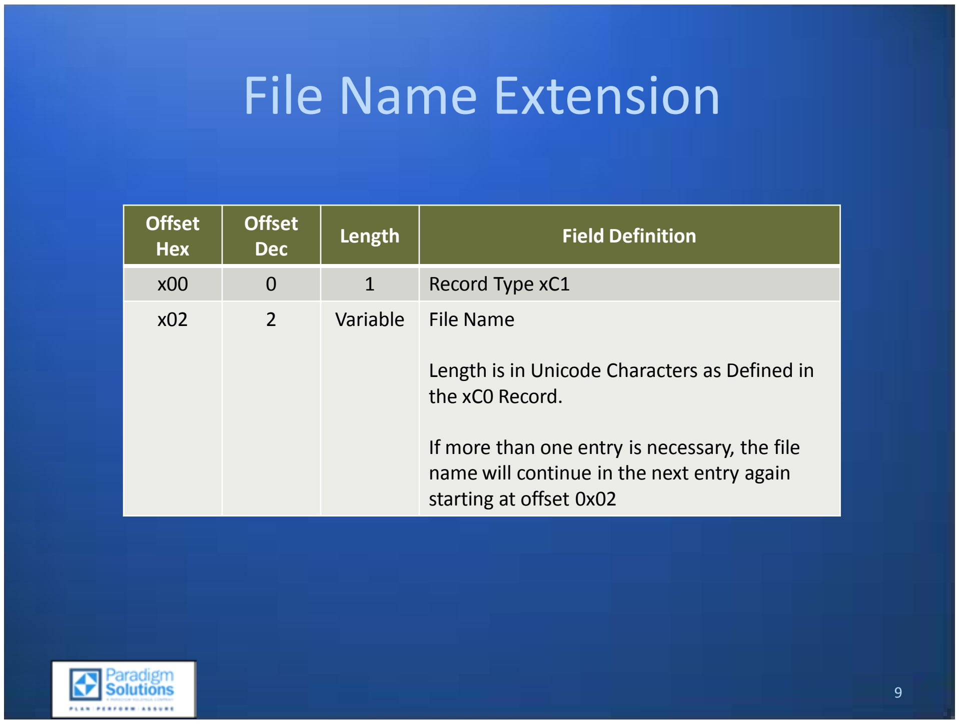
8

Stream Extension

Record type 0xC0 is the file metadata record for the directory entry. The metadata stored here includes secondary flags. The second byte is used to indicate whether the FAT chain is valid or not. This has not been tested thoroughly yet.

Offset Hex	Offset Dec	Length	Field Definition
x00	0	1	Record Type xC0
x01	1	1	Name Space Type (Similar to NTFS)
x02	2	2	Number of Unicode Characters in the File Name
x04	4	2	File Name Hash
x06	6	2	Unused
x08	8	8	Logical Size of the File in Bytes
x10	16	4	Unknown
x14	20	4	Starting Cluster of the File
x18	24	8	Logical Size of the File in Bytes

Table 5 - exFAT Stream Extension Data



File Name Extension

Offset Hex	Offset Dec	Length	Field Definition
x00	0	1	Record Type xC1
x02	2	Variable	File Name Length is in Unicode Characters as Defined in the xC0 Record. If more than one entry is necessary, the file name will continue in the next entry again starting at offset 0x02

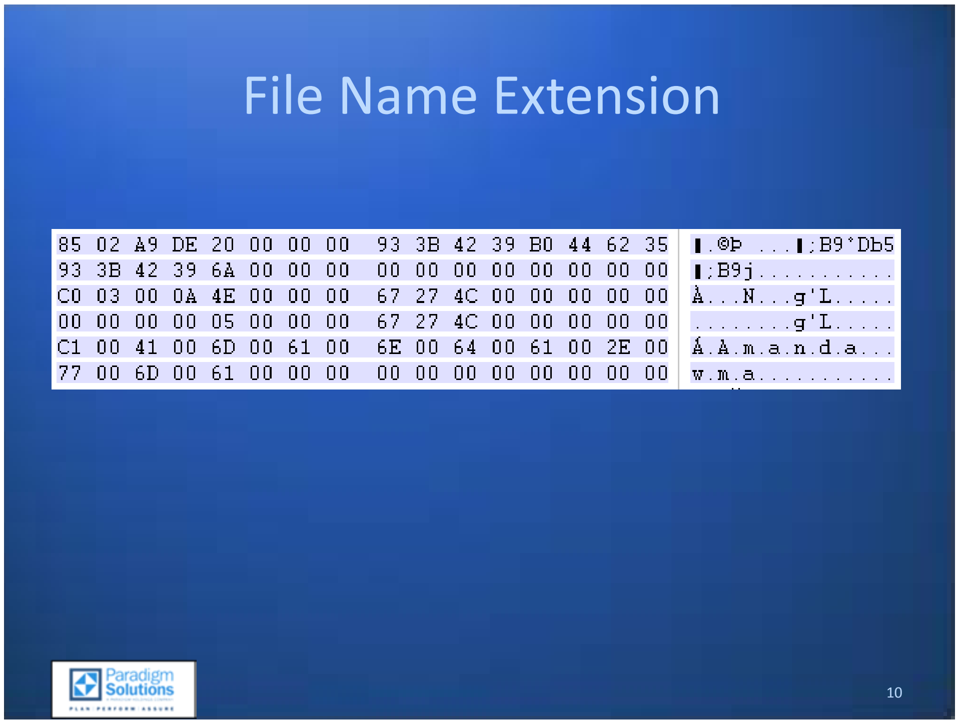
9

File Name Extension

The actual file name will be stored in the third (and subsequent) records. Each file name extension will start with 0xc100. The first Unicode character of the file name itself will be at offset 2. If more than 15 characters (30 bytes) are needed to store the file name, additional records will be generated to store the rest of the file name.

Offset Hex	Offset Dec	Length	Field Definition
x00	0	1	Record Type xC1
x02	2	Variable	File Name Length is in Unicode Characters as Defined in the xC0 Record. If more than one entry is necessary, the file name will continue in the next entry again starting at offset 0x02

Table 6 - exFAT File Name Extension Data



File Name Extension

```
85 02 A9 DE 20 00 00 00  93 3B 42 39 B0 44 62 35  ▌.©Þ ...▌;B9°Db5
93 3B 42 39 6A 00 00 00  00 00 00 00 00 00 00 00  ▌;B9j...........
C0 03 00 0A 4E 00 00 00  67 27 4C 00 00 00 00 00  À...N...g'L.....
00 00 00 00 05 00 00 00  67 27 4C 00 00 00 00 00  ........g'L.....
C1 00 41 00 6D 00 61 00  6E 00 64 00 61 00 2E 00  Á.A.m.a.n.d.a...
77 00 6D 00 61 00 00 00  00 00 00 00 00 00 00 00  w.m.a...........
```

10

File Name Extension

Using the provided offset data tables, this example can be parsed out manually. The file name is relatively short and the number of entries is minimal. The entry can be broken up into its three 32 byte records: directory entry record, stream extension, and the file name extension. All offsets start from the beginning of each individual record.

Directory entry record:

- Offset 0 – 0x85 – master record identifier
- Offset 1 – 0x02 – the number of entries required beyond the master record
- Offset 4 – 0x20000000 – DOS accessibility attributes (see Table 7 - exFAT DOS Attribute Flags)
- Offset 8 – 0x933B4239 – 32 bit created date and time
- Offset 12 – 0xB0446235 – 32 bit last access date and time
- Offset 16 – 0x933B4239 – 32 bit last modified date and time

Stream extension:

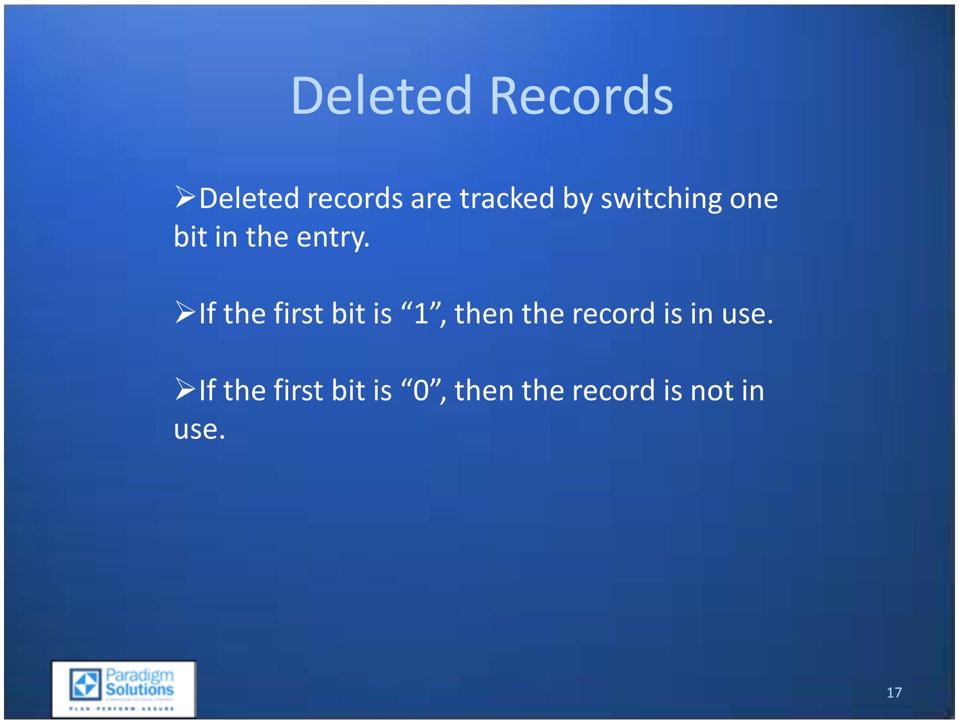
- Offset 0 – 0xC0 – file metadata record identifier
- Offset 1 – 0x03 – file name space identifier
- Offset 2 – 0x0A – Unicode characters in the file name (10 characters, 20 bytes)
- Offset 8 – 0x67274C00 – logical file size in bytes
- Offset 20 – 0x05000000 – starting logical cluster

File name extension:

- Offset 0 – 0xC1 – file name identifier
- Offset 2 – Amanda.wma – in Unicode, the file name

Hex	Binary	Description
0x0001	0000 0001	Read Only
0x0002	0000 0010	Hidden File
0x0004	0000 0100	System File
0x0020	0010 0000	Archive

Table 7 - exFAT DOS Attribute Flags



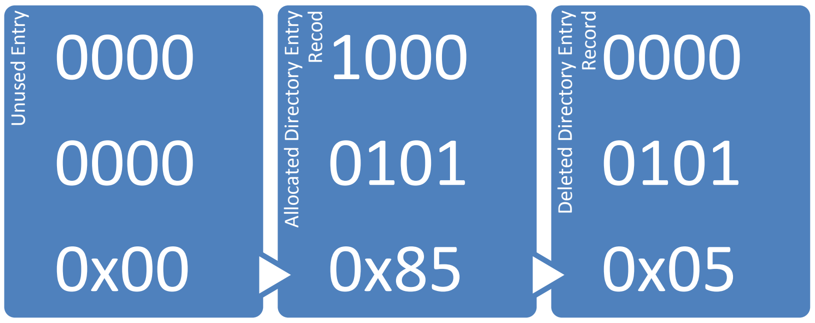
Deleted Records

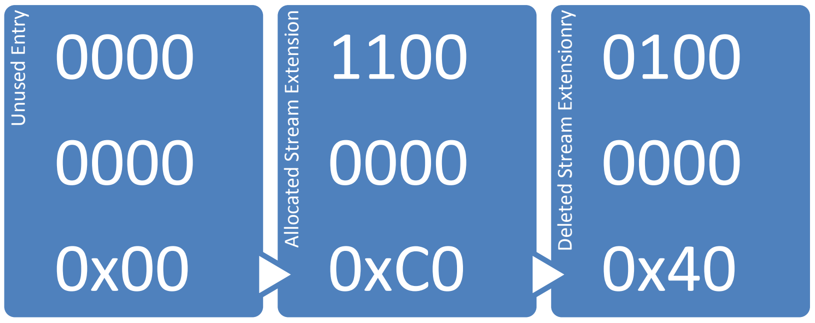
- Deleted records are tracked by switching one bit in the entry.
- If the first bit is “1”, then the record is in use.
- If the first bit is “0”, then the record is not in use.

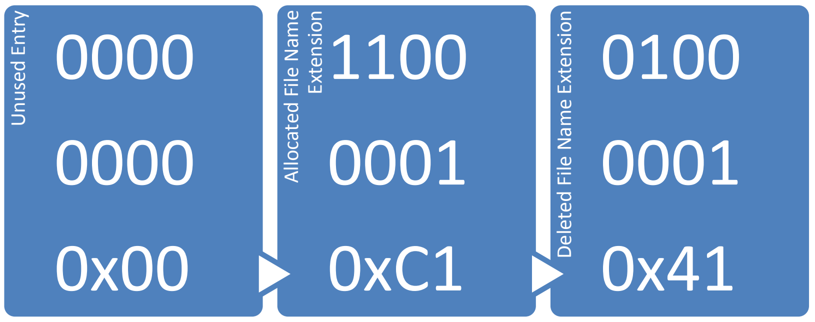
17

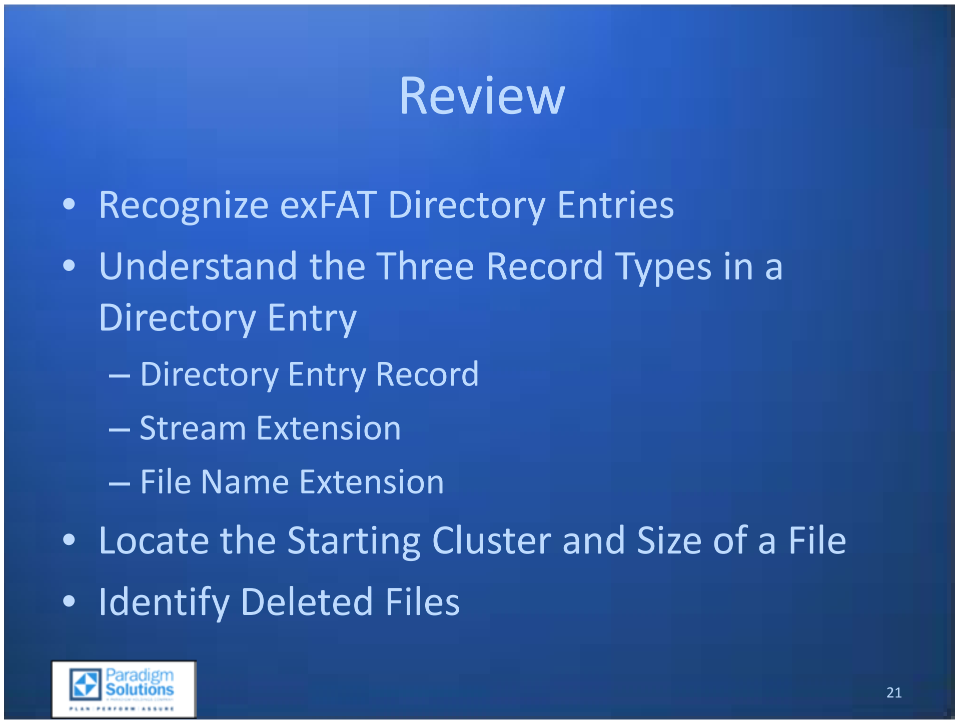
Deleted Records

The directory entry tracks deleted files by switching the first bit of each record type identifier from 1 to 0. This changes the displayed hex values in the record type identifier as listed in the figures below. There is an example of each byte broken down into the upper nibble, lower nibble, and the equivalent hex value.


Unused Entry
0000
0000
0x00
Allocated Directory Entry Recod
1000
0101
0x85
Deleted Directory Entry Record
0000
0101
0x05


Unused Entry
0000
0000
0x00
Allocated Stream Extension
1100
0000
0xC0
Deleted Stream Extensionry
0100
0000
0x40


Unused Entry
0000
0000
0x00
Allocated File Name Extension
1100
0001
0xC1
Deleted File Name Extension
0100
0001
0x41



Review

- Recognize exFAT Directory Entries
- Understand the Three Record Types in a Directory Entry
  - Directory Entry Record
  - Stream Extension
  - File Name Extension
- Locate the Starting Cluster and Size of a File
- Identify Deleted Files

Paradigm Solutions

21

Review

After completing this section, the examiner will now have the ability to:

- Recognize exFAT Directory Entries
- Understand the Three Record Types in a Directory Entry
  - o Directory Entry Record
  - o Stream Extension
  - o File Name Extension
- Locate the Starting Cluster and Size of a File
- Identify Deleted Files

Appendix I: exFAT Quick Reference

Shorthand	Longhand	*n*th	Bytes
Ki	Kilobyte	2^{10}	1024
Mi	Megabyte	2^{20}	1024 KiB
Gi	Gigabyte	2^{30}	1024 MiB
Ti	Terabyte	2^{40}	1024 GiB
Pi	Petabyte	2^{50}	1024 TiB
Ei	Exabyte	2^{60}	1024 PiB
Zi	Zetabyte	2^{70}	1024 EiB

Table 1 - Data Measurement Units

Offset Hex	Offset Dec	Length	Field Definition
x00	0	3	Jump Code
x03	3	8	OEM File System Identifier
x0B	11	35	Must be Zero
x40	64	4	Partition Sector Offset – Will be Zero for Removable Media
x48	72	8	Total Sectors on the Volume
x50	80	4	FAT Location in Sectors
x54	84	4	Physical Size of the FAT in Sectors
x58	88	4	Physical Sector Location of the Bitmap
x5C	92	4	Allocation Units on the Volume (Bit Count)
x60	96	4	1st Cluster of the Root Directory
x64	100	4	Volume Serial Number
x68	104	2	File System Revision Number – 1.0
X6A	106	1	Volume Flags
X6B	107	1	Active FAT
x6C	108	1	Bytes per Sector
x6D	109	1	Sectors Per Cluster (in Powers of 2)
x6E	110	1	The Number of FATs on the Volume
x70	112	1	Percentage In Use

Table 2 - exFAT Volume Boot Record

Offset Hex	Offset Dec	Length	Field Definition
x85	0	1	File Name Record, Master Entry
x83	1	1	Volume Name Record, Master Entry and Actual Volume Name
x82	2	2	Volume Name Record, Unknown Purpose
x81	4	2	Volume Name Record, Unknown Purpose
xC0	6	2	File Name Record, File System Metadata
xC1	8	4	File Name Record, Actual File Name

Table 3 – exFAT Directory Entry Record Type Values

Offset Hex	Offset Dec	Length	Field Definition
x00	0	1	Record Type x85 – File Record
x01	1	1	Number of 32 Byte Records in the Entry
x02	2	2	Unknown (Potential File Name Hash)?
x04	4	2	DOS File Flags (Archive, Hidden, etc)
x06	6	2	Unknown (Values only on Volume Label)
x08	8	4	Created Date and Time
x0C	12	4	Last Access Date and Time
x10	16	4	Last Modified Date and Time
x14	20	4	Unknown
x18	22	3	Time Zone Offset Applied to the File Time

Table 4 - exFAT Directory Entry Record Data

Offset Hex	Offset Dec	Length	Field Definition
x00	0	1	Record Type xC0
x01	1	1	Name Space Type (Similar to NTFS)
x02	2	2	Number of Unicode Characters in the File Name
x04	4	2	Unknown
x06	6	2	Unknown
x08	8	8	Logical Size of the File in Bytes
x10	16	4	Unknown
x14	20	4	Starting Cluster of the File
x18	24	8	Logical Size of the File in Bytes

Table 5 - exFAT Stream Extension Data

Offset Hex	Offset Dec	Length	Field Definition
x00	0	1	Record Type xC1
x02	2	Variable	File Name Length is in Unicode Characters as Defined in the xC0 Record. If more than one entry is necessary, the file name will continue in the next entry again starting at offset 0x02

Table 6 - exFAT File Name Extension Data

Hex	Binary	Description
0x0001	0000 0001	Read Only
0x0002	0000 0010	Hidden File
0x0004	0000 0100	System File
0x0020	0010 0000	Archive

Table 7 - exFAT DOS Attribute Flags

Appendix II: Sources and References

Carrier, B. (2005). File System Forensic Analysis. In B. Carrier, *File System Forensic Analysis.* Person Education, Inc.

Microsoft. (n.d.). *Description of the exFAT file system driver update package*. Retrieved September 2009, from Windows XP exFAT Package: http://support.microsoft.com/kb/955704

Microsoft. (n.d.). *Extended FAT File System*. Retrieved September 2009, from Microsoft Developers Network: http://msdn.microsoft.com/en-us/library/aa914353.aspx

Microsoft. (n.d.). *TFAT Overview*. Retrieved September 2009, from Microsoft Developers Network: http://msdn.microsoft.com/en-us/library/aa915463.aspx

Microsoft. (n.d.). *Windows History: Windows Desktop Products History*. Retrieved July 10, 2009, from Microsoft.com: www.microsoft.com/windows/WinHistoryDesktop.mspx

Parlante, N. (2001, April). *Linked List Basics*. Retrieved September 2009, from Stanford CS Education Library: http://cslibrary.stanford.edu/103/

Pudipeddi, R. V., Ghotge, V. V., & Thind, R. S. (2009). *Patent No. 20090164440.* US.

SD Association. (n.d.). Retrieved September 2009, from SD Card Association: http://www.sdcard.org/home/

Wikipedia. (2009, September). *exFAT*. Retrieved September 2009, from Wikipedia.org: http://en.wikipedia.org/wiki/ExFAT

Appendix III: Special Thanks

Independent Research

Myers, Jared
Computer Forensic Examiner
Contractor for the Department of Defense
Computer Forensics Laboratory
Linthicum, Maryland

Research and Theory Assistance

McCarthy, Nathan
Computer Forensic Analyst
Paradigm Solutions
Contractor for the US Department of State
Computer Investigations and Forensics
9715 Key West Avenue, Third Floor
Rockville, Maryland 20850

Stellatos, Gerry
Computer Forensic Analyst
Paradigm Solutions
Contractor for the US Department of State
Computer Investigations and Forensics
9715 Key West Avenue, Third Floor
Rockville, Maryland 20850

Witsman, Kristi
Computer Forensic Analyst
Formerly of Paradigm Solutions
9715 Key West Avenue, Third Floor
Rockville, Maryland 20850

Project Review

Beltz, Steve
Paradigm Solutions
9715 Key West Avenue, Third Floor
Rockville, Maryland 20850

Editorial Review

Stedman, Karen
Technician
Contractor for the US Department of State
Computer Investigations and Forensics
9715 Key West Avenue, Third Floor
Rockville, Maryland 20850

Schohn, Steve
Computer Forensic Analyst
Paradigm Solutions
Contractor for the US Department of State
Computer Investigations and Forensics
9715 Key West Avenue, Third Floor
Rockville, Maryland 20850