Windows Imaging File Format (WIM)

Last Updated: October 2007

Applies to: Software Developers

Abstract: This paper defines the internal format of a Windows Imaging (WIM) file format. This information may be used to build .wim file creation or extraction tools, or other WIM-enabled applications.

This information applies for the following operating systems:

Windows Vista™

Disclaimer

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AS TO THE INFORMATION IN THIS DOCUMENT.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred.

Microsoft does not make any representation or warranty regarding specifications in this document or any product or item developed based on these specifications. Microsoft disclaims all express and implied warranties, including but not limited to the implied warranties or merchantability, fitness for a particular purpose and freedom from infringement. Without limiting the generality of the foregoing, Microsoft does not make any warranty of any kind that any item developed based on these specifications, or any portion of a specification, will not infringe any copyright, patent, trade secret or other intellectual property right of any person or entity in any country. It is your responsibility to seek licenses for such intellectual property rights where appropriate. Microsoft shall not be liable for any damages arising out of or in connection with the use of these specifications, including liability for lost profit, business interruption, or any other damages whatsoever. Some states do not allow the exclusion or limitation of liability or consequential or incidental damages; the above limitation may not apply to you.

Contents

Introduction

This paper defines the Microsoft® Windows® Imaging file format (WIM). WIM is a file-based disk image format introduced in Windows Vista. WIM files are compressed packages containing a number of related files. The format of a WIM file is optimized for maximum compression using LZX, fast compression using XPRESS, or uncompressed. This document does not define these compression formats. The purpose of this document is not to enable developers to encode or decode WIM-compressed data.

The primary tools for creating and managing a WIM file are ImageX and the Windows Imaging Interface Reference.

ImageX—A command-line tool that works with the latest Microsoft Windows image (.wim) files. The .wim files contain one or more volume images for a Windows operating system, while a volume image represents the captured volume or partition of a Windows operating system. The primary purpose of ImageX is to capture, modify, and apply images for deployment in a manufacturing or corporate IT environment.

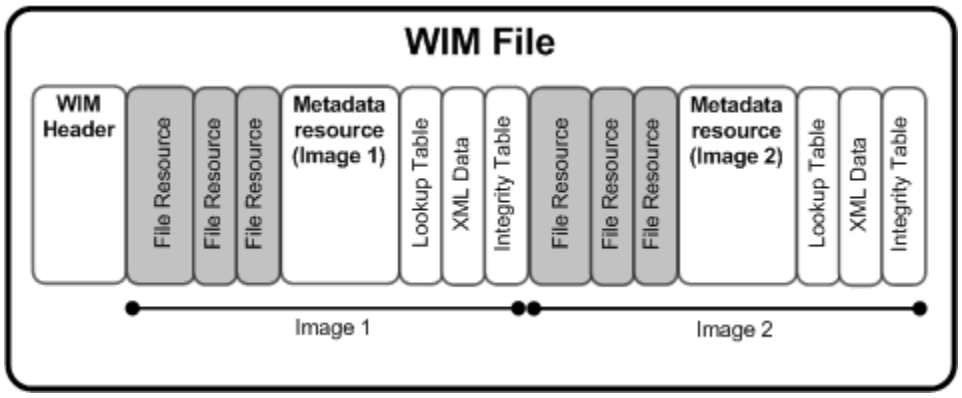
Typically, you'll use ImageX with the Windows Imaging File System Filter (WIM FS Filter) driver. The WIM FS Filter driver enables you to mount an image to a directory, where you can browse, copy, paste, and edit the files from a file-management tool, such as Windows Explorer, without extracting or re-creating the image. ImageX is commonly used in a Windows PE environment during image-based deployments.

Windows Imaging Interface Reference—Describes the programmatic method for managing Windows image (.wim) files. The primary purpose of Imaging APIs (Wimgapi.dll) is to programmatically capture, modify, and apply images for deployment in a manufacturing or corporate IT environment. ImageX is an implementation of the Imaging APIs.

ImageX and Window Imaging Interface Reference are available through the Windows OEM Preinstallation Kit (Windows OPK) or Windows Automated Installation Kit (Windows AIK) (http://go.microsoft.com/fwlink/?LinkId=86612).Windows OPK is available only to Original Equipment Manufacturers (OEMs) and system builders. Contact your local Microsoft representative.

WIM File Structure

A WIM file structure contains up to six types of resources: header, file resource, metadata resource, lookup table, XML data, and integrity table. The following illustration shows the general layout of a WIM file that contains two images.



WIM File Layout

- **WIM Header**—Defines the content of the .wim file, including memory location of key resources (metadata resource, lookup table, XML data), and various .wim file attributes (version, size, compression type).
- **File Resources**—Series of packages that contain captured data, such as source files.
- **Metadata Resource**—Contains information about the files you are capturing, including directory structure and file attributes. There is one metadata resource for each image in a .wim file.
- **Lookup Table**—Contains memory location of file resource files in the .wim file.
- **XML Data**—Contains additional data about the image.
- **Integrity Table**—Contains security hash information used to verify an image's integrity during an apply operation.

Image Capture Process

The following steps describe the process of how a .wim file is created by using the ImageX /capture command. For example, **imagex /capture c:\mydata c:\data.wim "My Data"**

1. A WIM Header is created on disk (c:\data.wim) with initial values.
2. A metadata resource is created in memory and the content of c:\mydata is scanned and indexed.
3. Using the metadata resource, the content of c:\mydata is written into 32K resource files chunks to the disk. At the same time, the memory location of each chunk is written to a lookup table in memory.
4. When all data is captured, the metadata resource, lookup table, and XML data are written to disk.
5. The WIM header is updated.
6. The capture is completed.

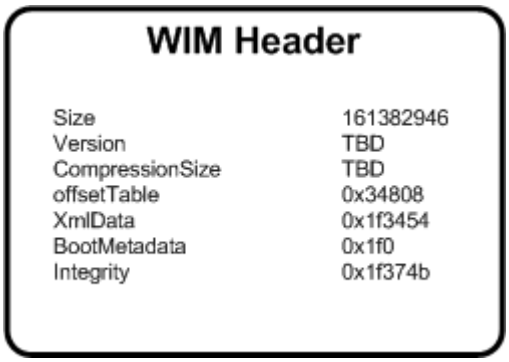
Image Apply Process

The following steps describe the process of how data from a .wim file is extracted by using the ImageX /apply command. For example, **imagex /apply c:\data.wim 1 d:**

1. The WIM header is opened, and the location of metadata resource is found.
2. The metadata resource is loaded into memory.
3. The metadata resource processes the lookup table for the location of file resources.
4. The directory structure is created.
5. Files are written to the directory structure, and file attributes are applied.
6. Repeat the previous step until all files are written out to the destination.

WIM Header

The WIM header stores general information about the .wim file, including how many images are in the file, the size of each image, the compression type, and memory location of resource files. The following illustration shows the type of information stored in a WIM header.



WIM Header

```
typedef struct _WIMHEADER_V1_PACKED
{
   CHAR               ImageTag[8];        // "MSWIM\0\0"
   DWORD              cbSize;
   DWORD              dwVersion;
   DWORD              dwFlags;
   DWORD              dwCompressionSize;
   GUID               gWIMGuid;
   USHORT             usPartNumber;
   USHORT             usTotalParts;
   DWORD              dwImageCount;
   RESHDR_DISK_SHORT rhOffsetTable;
   RESHDR_DISK_SHORT rhXmlData;
   RESHDR_DISK_SHORT rhBootMetadata;
   DWORD              dwBootIndex;
   RESHDR_DISK_SHORT rhIntegrity;
   BYTE               bUnused[60];
}
WIMHEADER_V1_PACKED, *LPWIMHEADER_V1_PACKED;
```

ImageTag

Signature that identifies the file as a .wim file. Value is set to “MSWIM\0\0”.

Size

Size of the WIM header in bytes.

Version

The current version of the .wim file. This number will increase if the format of the .wim file changes.

Flags

Defines the following custom flags.

```
#define FLAG_HEADER_RESERVED            0x00000001
#define FLAG_HEADER_COMPRESSION         0x00000002
#define FLAG_HEADER_READONLY            0x00000004
#define FLAG_HEADER_SPANNED             0x00000008
#define FLAG_HEADER_RESOURCE_ONLY       0x00000010
#define FLAG_HEADER_METADATA_ONLY       0x00000020
#define FLAG_HEADER_WRITE_IN_PROGRESS   0x00000040
#define FLAG_HEADER_RP_FIX              0x00000080 // reparse point fixup
#define FLAG_HEADER_COMPRESS_RESERVED   0x00010000
#define FLAG_HEADER_COMPRESS_XPRESS     0x00020000
#define FLAG_HEADER_COMPRESS_LZX        0x00040000
```

Flag	Meaning
FLAG_HEADER_COMPRESSION	Resources within the WIM (both file and metadata) are compressed.
FLAG_HEADER_READONLY	The contents of this WIM should not be changed.
FLAG_HEADER_SPANNED	Resource data specified by the images within this WIM may be contained in another WIM.
FLAG_HEADER_RESOURCE_ONLY	This WIM contains file resources only. It does not contain any file metadata.
FLAG_HEADER_METADATA_ONLY	This WIM contains file metadata only.
FLAG_HEADER_WRITE_IN_PROGRESS	Limits one writer to the WIM file when opened with the WIM_FLAG_SHARE_WRITE mode, This flag is primarily used in the Windows Deployment Services (WDS) scenario.

Additionally, if the **FLAG_HEADER_COMPRESSION** flag is set, the following flags are valid:

Flag	Meaning
FLAG_HEADER_COMPRESS_XPRESS	Resources within the wim are compressed using XPRESS compression.
FLAG_HEADER_COMPRESS_LZX	Resources within the wim are compressed using LZX compression.

CompressionSize

Size of the compressed .wim file in bytes.

WIMGuid

A unique identifier.

PartNumber

The part number of the current .wim file in a spanned set. This value is **1**, unless the data of the .wim file was split into multiple parts (.swm).

TotalParts

The total number of .wim file parts in a spanned set.

ImageCount

The number of images contained in the .wim file.

OffsetTable

The location of the resource lookup table.

XmlData

The location of the XML data.

BootMetadata

The location of the metadata resource.

BootIndex

The index of the bootable image in the .wim file. If this is zero, then there are no bootable images available.

Integrity

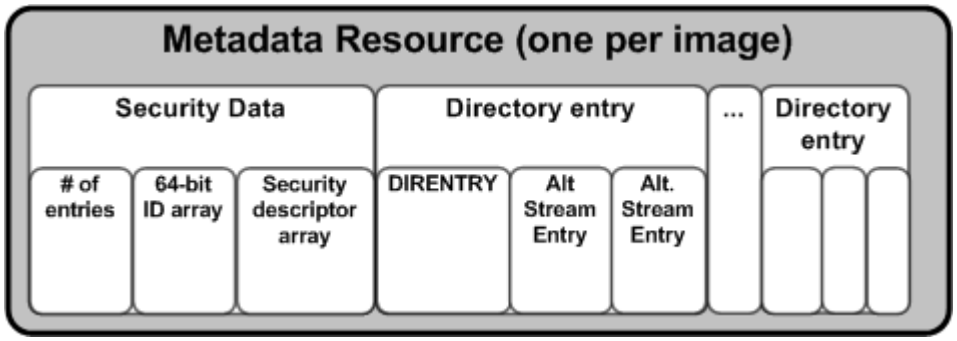
The location of integrity table used to verify files.

Unused

A reserved 60 bytes of additional space for future fields.

WIM Metadata Resource

The WIM metadata resource stores information on how captured data is organized in the .wim file. The metadata resource data is used to re-assemble source files back into its original directory structure. There is one metadata resource for each captured image in a .wim file. A metadata resource contains two sets of data: security data and directory data. The following illustration shows an example metadata resource layout.



WIM Metadata Resource Layout

Security Data

The first part of a metadata resource structure is a security block, the SECURITYBLOCK_DISK

structure, which describes the number and sizes of single-instanced security descriptors in the image.

```
typedef struct _SECURITYBLOCK_DISK
{
    DWORD           dwTotalLength;
    DWORD           dwNumEntries;
    ULARGE_INTEGER  liEntryLength[0];
}
```

TotalLength

The total length of the table.

NumEntries

The number of directory entries.

EntryLength

An array of sizes of the actual security descriptors to follow, in bytes.

The root directory entry for the image directory structure is located on a quad-aligned address after the security table. The structure is primarily constructed of DIRENTRY structures described below.

Directory Entry

The second part of a metadata resource structure is a root directory entry for the image directory structure. A directory entry is defined by the DIRENTRY structure.

```
typedef struct _DIRENTRY
{
    LARGE_INTEGER    liLength;
    DWORD            dwAttributes;
    DWORD            dwSecurityId;
    LARGE_INTEGER    liSubdirOffset;
    LARGE_INTEGER    liUnused1;
    LARGE_INTEGER    liUnused2;
    LARGE_INTEGER    liCreationTime;
    LARGE_INTEGER    liLastAccessTime;
    LARGE_INTEGER    liLastWriteTime;
    BYTE             bHash[HASH_SIZE];
    DWORD            dwReparseTag;
    DWORD            dwReparseReserved;
    LARGE_INTEGER    liHardLink;
    USHORT           wStreams;
    USHORT           wShortNameLength;
    USHORT           wFileNameLength;
    WCHAR            FileName[0];
}
```

liLength

Size of directory entry in bytes.

dwAttributes

The file attributes associated with this file.

dwSecurityId

The index of the node in the security table that contains this file's security information. If this field is set to -1, no security information exists for this file.

liSubdirOffset
The offset, from the start of the metadata section, of this directory entry's child files.

liUnused1
Reserved field.

liUnused2
Reserved field.

liCreationTime
The timestamp when entry was made.

liLastAccessTime
The timestamp when entry was last accessed.

liLastWriteTime
The timestamp when entry was last updated.

bHash[hash size]
A hash of the file's contents that uniquely identifies the contents in the hash table.

dwReparseTag
Identity of a reparse point. For example, this identifies the reparse point as a symbolic link, a mount point, or some third-party reparse point type. The data contained in the reparse point is interpreted differently by NTFS or file system filter drivers depending on what its identity is.

dwReparseReserved
Reserved field.

liHardLink
If this file is part of a hard link set, all the directory entries in the set will share the same value in this field.

wStreams
The number of STREAMENTRY structures that follow this DIRENTRY structure.

wShortNameLength
The size of the short filename in bytes.

wFileNameLength
The size of the filename in bytes.

FileName
The relative name of the file or directory.

The filename in this structure is the relative name. To determine the full path, you must reconstruct the path by traversing the parents of a particular DIRENTRY. There are several unions used, so the boot drivers can overlay a field with a pointer, rather than allocating a new copy of the metadata resource or having to recalculate locations of the parent directory or subdirectories.

Alternate Stream Entry

If the file or directory has an alternate file stream, then there are one or more STREAMENTRY structures that immediately follow the DIRENTRY.

```
typedef struct _STREAMENTRY
{
     LARGE_INTEGER      liLength;
     LARGE_INTEGER      Unused1;
     BYTE               bHash[HASH_SIZE];
     USHORT             wStreamNameLength;
     WCHAR              StreamName[0];
}
```

liLength

The size of the alternate stream entry in bytes.

Unused1

A reserved field.

bHash[HASH_SIZE]

A SHA-1 hash of the alternate stream entry.

wStreamNameLength

The size of the stream name in bytes.

StreamName[0]

The name of the alternate stream.

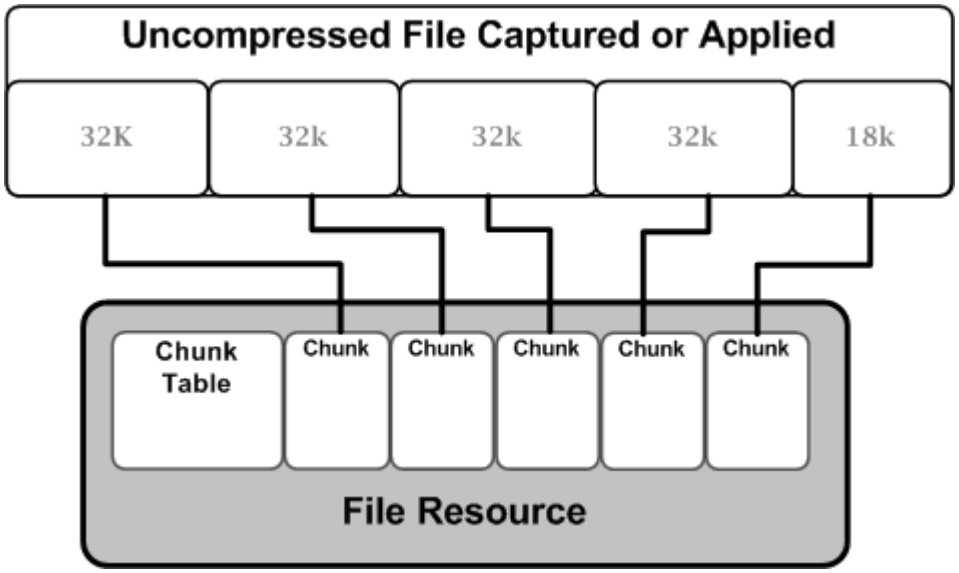
The file resource lookup for DIRENTRY and STREAMENTRY is done by the bHash field, which corresponds to the bHash field in the lookup table. To extract the resource for a particular file, a match on this hash must be found.

In addition:

- Security descriptors are single-instanced within an image.
- Each directory, not directory entry, ends with a 64-bit zero to mark the end of a directory.
- Each directory entry is four-byte aligned in order to align with each directory.

WIM File Resource

This section describes the structure of a WIM file resource. A file resource is a container that stores compressed source files. When data is captured into WIM, the data is stored into 32K chunks and then compressed into smaller chunks. The location of each chunk is stored in a chunk table within the file resource structure.



File Resource Compression Process

A file resource entry is of type RESHDR_DISK_SHORT structure.

```
typedef struct _RESHDR_BASE_DISK
{
    ULONGLONG ullSize;
    BYTE  sizebytes[7];
    LARGE_INTEGER liOffset;
}

typedef struct _RESHDR_DISK_SHORT
{
    RESHDR_BASE_DISK        Base;  // Must be first.
    LARGE_INTEGER           liOriginalSize;
}
```

ullSize (bFlags)

One-byte reserved flag. Flags include:

```
#define RESHDR_FLAG_FREE            0x01
#define RESHDR_FLAG_METADATA        0x02
#define RESHDR_FLAG_COMPRESSED      0x04
#define RESHDR_FLAG_SPANNED         0x08
```

sizebytes[7]

The compressed size of the file.

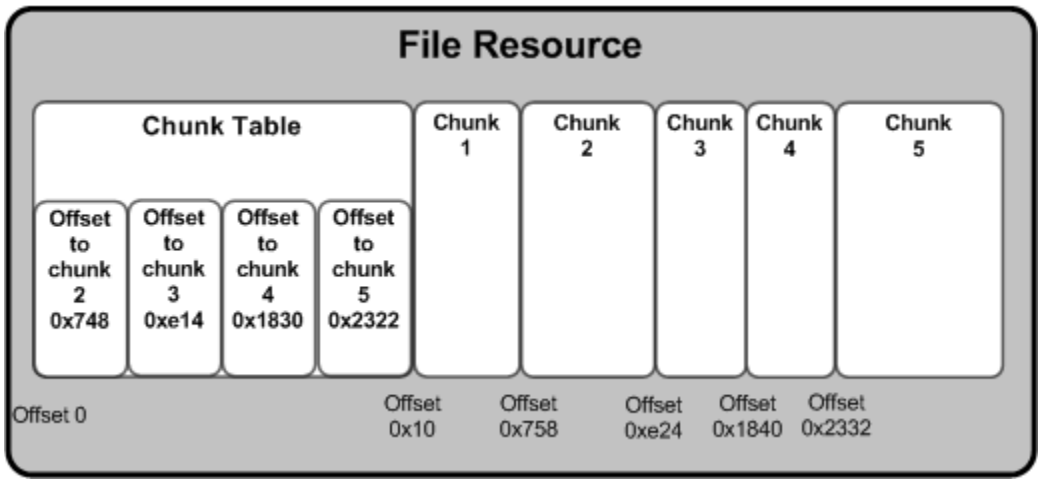
liOffset

The location of the resource within the .wim file.

liOriginalSize

The original uncompressed size of the file.

The following illustration shows how captured data is divided and then stored into a file resource structure.



File Resource Layout

In a compressed file resource, the chunk table is placed at the beginning of the resource. An uncompressed file is divided into 32K block sizes, compressed and then stored sequentially after the chunk table.

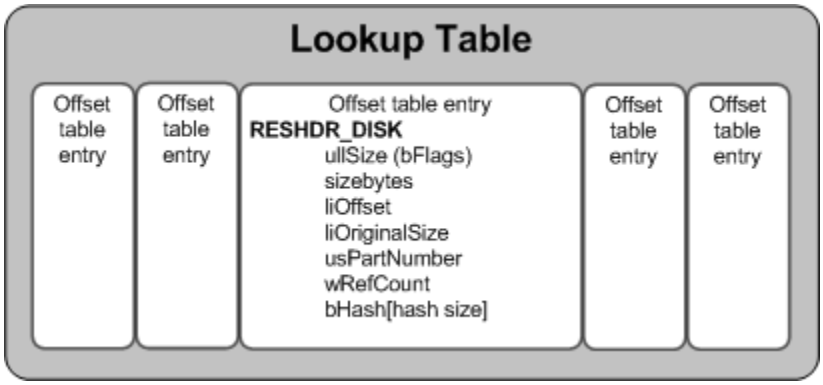
The number of chunk entries will be

```
( ( ( liOriginalSize.QuadPart + 32768 – 1 ) / 32768 ) – 1 )
```

If the original file is >4GB, then the chunk entry is a LARGE_INTEGER; otherwise it is a DWORD. Each value is the offset of the next chunk, so you can compute what the block compressed to. For example, if the entries are 0x4000, 0x6000, 0x12000, then you know that block #1 compressed to 0x4000 bytes, block #2 compressed to 0x2000 bytes, and block #3 compressed to 0x6000 bytes. All three should expand to 32768, unless it is the end block, which will uncompress to whatever is left over. The block decompressor is determined by the WIM header's flag value.

Lookup Table

This section describes the structure of the resource lookup table. The lookup table is an array of RESHDR_DISK structures. The lookup table provides the location and size of the resources within the .wim file. In addition, it provides part number, reference count, and a SHA-1 hash of the uncompressed data. The following is a general layout of the lookup table.



Lookup Table Layout

```
typedef struct _RESHDR_DISK
{
    RESHDR_DISK_SHORT;
    USHORT                  usPartNumber;
    DWORD                   dwRefCount;
    BYTE                    bHash[HASH_SIZE];
}
```

ullSize (bFlags)

One-byte reserved flag. Flags include:

```
#define RESHDR_FLAG_FREE            0x01
#define RESHDR_FLAG_METADATA        0x02
#define RESHDR_FLAG_COMPRESSED      0x04
#define RESHDR_FLAG_SPANNED         0x08
```

sizebytes[7]

The compressed size of the file.

liOffset

The location of the resource within the .wim file.

liOriginalSize

The original size of the file.

usPartNumber

The part number in a spanned set.

dwRefCount

The total number of times this file resource is stored in all images within a single .wim file.

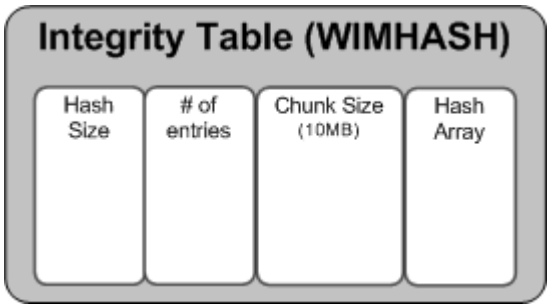
bHash[HASH_SIZE]

The SHA-1 hash of the uncompressed data.

To find images within a WIM file, you can search the lookup table for resources with the REDHDR_FLAG_METADATA flag set.

Integrity Table

This section describes the structure of the integrity table resource. This resource is optional and is created when you set the /check switch during an ImageX capture operation. When created, the resource contains the following parts.



Integrity Table Layout

```
typedef struct _WIMHASH
```

```
{
    DWORD cbSize;
    DWORD dwNumElements;
    DWORD dwChunkSize;
    BYTE  abHashList[0];
}
```

cbSize

The size of the entire integrity table resource in bytes.

dwNumElements

The number of entries in the table.

dwChunkSize

The size of each hash chunk in bytes.

abHashList

The list of entries.

In addition:

- Each chunk entry uses a 20-byte SHA-1 hash algorithm.
- The calculation of each chunk begins at the end of the WIM header to the end of the lookup table. Currently the chunk size is fixed at 10MB.

XML Data

This section describes the structure of the xml data resource. This resource is an uncompressed clear text XML file that is stored after each lookup table resource. The file begins with a Unicode lead byte of 0xFEFF followed by the following default XML structure.

```
<WIM>
  <TOTALBYTES>121960277</TOTALBYTES>
  <IMAGE INDEX="1">
    <DIRCOUNT>526</DIRCOUNT>
    <FILECOUNT>3030</FILECOUNT>
    <TOTALBYTES>258581030</TOTALBYTES>
    <CREATIONTIME>
      <HIGHPART>0x01C6FE73</HIGHPART>
      <LOWPART>0x4ADB2D90</LOWPART>
    </CREATIONTIME>
    <LASTMODIFICATIONTIME>
      <HIGHPART>0x01C6FE73</HIGHPART>
      <LOWPART>0x5E8DBB1E</LOWPART>
    </LASTMODIFICATIONTIME>
  </IMAGE>
</WIM>
```

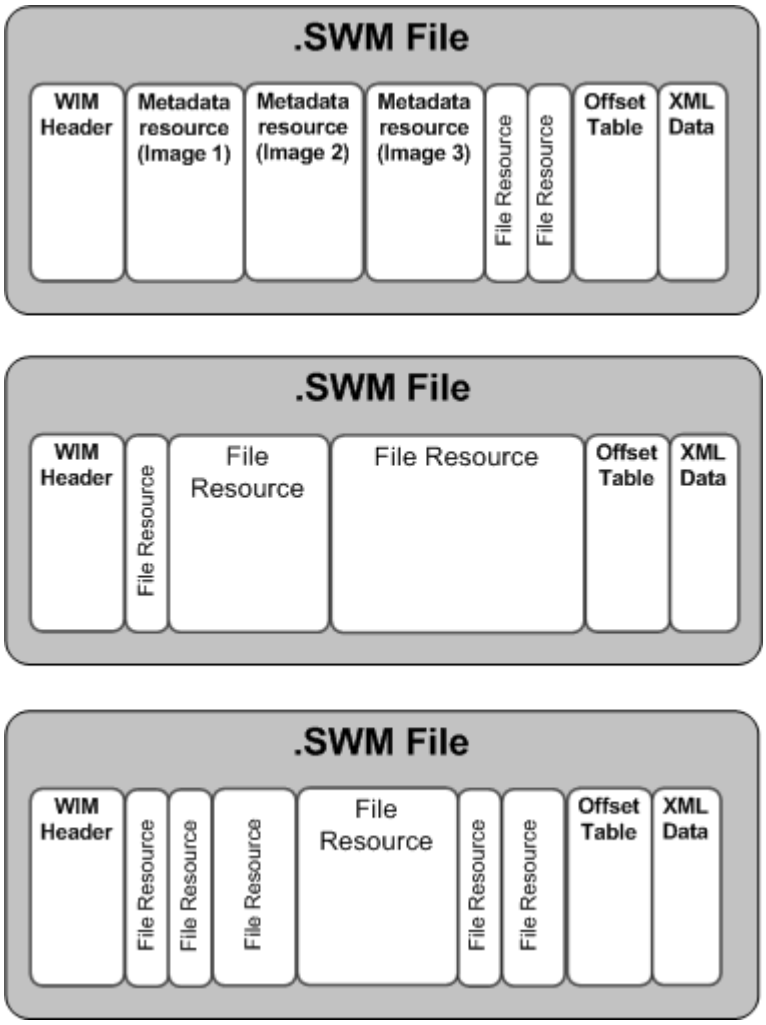
In addition:

- ImageX captures additional information including <NAME>, <FLAGS>, <DESCRIPTION>, and a <WINDOWS> structure to support Windows Vista Setup functionality.
- ImageX /info command displays information based on this XML data resource.

Split (Spanned) WIM

This section describes the structure of a split WIM file. A WIM can be split (spanned) into multiple parts called .swm files. When a WIM is split, the first part will always contain a copy of the WIM

header and all metadata resources. Subsequent .swm files will contain the remaining resources (file resources, lookup table, xml data, and integrity table). The following illustration shows the layout of a split WIM that originally contained three images.



Sample Split WIM layout