

branch: master ▾

RawDir / RawDir.au3



jschicht 2 months ago Version 1.0.0.0

1 contributor

file 1996 lines (1920 sloc) 92.539 kb

Open Edit Raw Blame History Delete

```
1 #RequireAdmin
2 #Region ;**** Directives created by AutoIt3Wrapper_GUI ****
3 #AutoIt3Wrapper_Change2CUI=y
4 #AutoIt3Wrapper_Res_Comment=Raw directory listing
5 #AutoIt3Wrapper_Res_Description=Low level dir command
6 #AutoIt3Wrapper_Res_Fileversion=1.0.0.0
7 #AutoIt3Wrapper_Res_LegalCopyright=Joakim Schicht
8 #AutoIt3Wrapper_Res_requestedExecutionLevel=asInvoker
9 #EndRegion ;**** Directives created by AutoIt3Wrapper_GUI ****
10 #Include <WinAPIEx.au3>
11 #include <Array.au3>
12 #Include <String.au3>
13 #Include <FileConstants.au3>
14
15 Global $DirArray,$NeedIndx=0, $ResidentIndx, $AttributesArr[18][4], $DoExtractMeta=False, $TargetFileName, $DATA_Name, $FN_
16 Global $TargetImageFile, $Entries, $InputFile, $IsShadowCopy=False, $IsPhysicalDrive=False, $IsImage=False, $hDisk, $sBuffer
17 Global $OutPutPath=@ScriptDir, $InitState = False, $DATA_Clusters, $AttributeOutFileName, $DATA_InitSize, $ImageOffset, $AD
18 Global $TargetDrive = "", $ALInnerCouner, $MFTSize, $TargetOffset, $SectorsPerCluster,$MFT_Record_Size,$BytesPerCluster,$By
19 Global $IsolatedAttributelist, $AttribListNonResident=0,$IsCompressed,$IsSparse, $_COMMON_KERNEL32DLL=DllOpen("kernel32.dll
20 Global $RUN_VCN[1],$RUN_Clusters[1],$MFT_RUN_Clusters[1],$MFT_RUN_VCN[1],$DataQ[1],$AttribX[1],$AttribXType[1],$AttribXCoun
21 Global $IndxEntryNumberArr[1],$IndxMFTReferenceArr[1],$IndxMFTRefSeqNoArr[1],$IndxIndexFlagsArr[1],$IndxMFTReferenceOfParen
22 Global $IRArr[12][2],$IndxArr[20][2]
23 Global $DateTimeFormat = 6 ; YYYY-MM-DD HH:MM:SS:MSMSMS:NSNSNSNS = 2007-08-18 08:15:37:733:1234
24 Global $tDelta = _WinTime_GetUTCToLocalFileTimeDelta()
25 Global Const $RecordSignature = '46494C45' ; FILE signature
26 Global Const $RecordSignatureBad = '44414142' ; BAAD signature
27 Global Const $STANDARD_INFORMATION = '10000000'
28 Global Const $ATTRIBUTE_LIST = '20000000'
29 Global Const $FILE_NAME = '30000000'
30 Global Const $OBJECT_ID = '40000000'
31 Global Const $SECURITY_DESCRIPTOR = '50000000'
32 Global Const $VOLUME_NAME = '60000000'
33
34 Global Const $VOLUME_INFORMATION = '70000000'
35 Global Const $DATA = '80000000'
36 Global Const $INDEX_ROOT = '90000000'
37 Global Const $INDEX_ALLOCATION = 'A0000000'
38 Global Const $BITMAP = 'B0000000'
39 Global Const $REPARSE_POINT = 'C0000000'
40 Global Const $EA_INFORMATION = 'D0000000'
41 Global Const $EA = 'E0000000'
42 Global Const $PROPERTY_SET = 'F0000000'
43 Global Const $LOGGED_UTILITY_STREAM = '00010000'
44 Global Const $ATTRIBUTE_END_MARKER = 'FFFFFFF'
45 Global Const $FileInternalInformation = 6
46 Global Const $OBJ_CASE_INSENSITIVE = 0x00000040
47 Global Const $FILE_DIRECTORY_FILE = 0x00000002
48 Global Const $FILE_NON_DIRECTORY_FILE = 0x00000040
49 Global Const $FILE_RANDOM_ACCESS = 0x00000800
50 Global Const $tagIOSTATUSBLOCK = "dword Status;ptr Information"
51 Global Const $tagOBJECTATTRIBUTES = "ulong Length;hwnd RootDirectory;ptr ObjectName;ulong Attributes;ptr SecurityDescriptor"
52 Global Const $tagUNICODESTRING = "ushort Length;ushort MaximumLength;ptr Buffer"
53 Global Const $tagFILEINTERNALINFORMATION = "int IndexNumber;"
54 Global $Timerstart = TimerInit()
55
56 ConsoleWrite("RawDir v1.0.0.0" & @CRLF)
57
```

```

58 If $cmdline[0] <> 2 Then
59     ConsoleWrite("Error: Wrong input" & @CRLF)
60     _Usage()
61     Exit
62 ElseIf $cmdline[1] <> 1 And $cmdline[1] <> 2 Then
63     ConsoleWrite("Error: Wrong param1: " & $cmdline[1] & @CRLF)
64     _Usage()
65     Exit
66 ElseIf DriveGetFileSystem(StringMid($cmdline[2],1,2)&"\") <> "NTFS" Then
67     ConsoleWrite("Error: Target drive is not valid or is not a NTFS volume: " & $cmdline[2] & @CRLF)
68     Exit
69 EndIf
70 $DetailMode = $cmdline[1]
71 $StartStr = $cmdline[2]
72 If StringLen($StartStr)=2 Then $StartStr&="\\"
73 $TargetDrive = StringMid($StartStr,1,2)
74 $ParentDir = _GenDirArray($StartStr)
75 Global $MftRefArray[$DirArray[0]+1]
76 _ReadBootSector($TargetDrive)
77 $BytesPerCluster = $SectorsPerCluster*$BytesPerSector
78 $MFTEntry = _FindMFT(0)
79 _DecodeMFTRecord($MFTEntry,0)
80 _DecodeDataQEntry($DataQ[1])
81 $MFTSize = $DATA_RealSize
82 Global $RUN_VCN[1], $RUN_Clusters[1]
83 _ExtractDataRuns()

84 $MFT_RUN_VCN = $RUN_VCN
85 $MFT_RUN_Clusters = $RUN_Clusters
86
87 $hDisk = _WinAPI_CreateFile("\\.\\" & $TargetDrive,2,2,7)
88 If $hDisk = 0 Then
89     ConsoleWrite("CreateFile: " & _WinAPI_GetLastError() & @CRLF)
90     Exit
91 EndIf
92
93 $NextRef = 5
94 $MftRefArray[1]=$NextRef
95 $ResolvedPath = $DirArray[1]
96 For $i = 2 To $DirArray[0]
97     Global $DataQ[1],$AttribX[1],$AttribXType[1],$AttribXCounter[1]
98     $NewRecord = _FindFileMFTRecord($NextRef)
99     _DecodeMFTRecord($NewRecord,1)
100    $NextRef = _ParseIndex($DirArray[$i])
101    $MftRefArray[$i]=$NextRef
102    If @error Then
103        Global $DataQ[1],$AttribX[1],$AttribXType[1],$AttribXCounter[1]
104        $NewRecord = _FindFileMFTRecord($MftRefArray[$i-1])
105        _DecodeMFTRecord($NewRecord,1)
106        $LastCheck = _DisplayList($ResolvedPath)
107    ElseIf $i=$DirArray[0] Then
108        Global $DataQ[1],$AttribX[1],$AttribXType[1],$AttribXCounter[1]
109        $NewRecord = _FindFileMFTRecord($MftRefArray[$i])
110        _DecodeMFTRecord($NewRecord,1)
111        $LastCheck = _DisplayList($ResolvedPath & "\" & $DirArray[$i])
112        If @error Then ; In case last part was a file and not a directory
113            Global $DataQ[1],$AttribX[1],$AttribXType[1],$AttribXCounter[1]
114            $NewRecord = _FindFileMFTRecord($MftRefArray[$i-1])
115            _DecodeMFTRecord($NewRecord,1)
116            $LastCheck = _DisplayList($ResolvedPath)
117        EndIf
118    ElseIf StringIsDigit($NextRef) Then
119        $ResolvedPath &= "\" & $DirArray[$i]
120        ContinueLoop
121    Else
122        ConsoleWrite("Error: Something went wrong" & @CRLF)
123        ExitLoop
124    EndIf
125 Next
126 ConsoleWrite(@CRLF)
127 _End($Timerstart)
128 Exit
129
130 Func _DisplayList($DirListPath)
131     If $DetailMode = 1 Then

```

```

132 If $AttributesArr[10][2] = "TRUE" Then; $INDEX_ALLOCATION
133     ConsoleWrite("Directory listing for: " & $DirListPath & @CRLF & @CRLF)
134
135     For $j = 1 To Ubound($IndxFileNameArr)-1
136         ConsoleWrite("Entry number: " & $IndxEntryNumberArr[$j] & @CRLF)
137         ConsoleWrite("FileName: " & $IndxFileNameArr[$j] & @CRLF)
138         ConsoleWrite("MFT Ref: " & $IndxMFTReferenceArr[$j] & @CRLF)
139         ConsoleWrite("MFT Ref SeqNo: " & $IndxMFTRefSeqNoArr[$j] & @CRLF)
140         ConsoleWrite("Parent MFT Ref: " & $IndxMFTReferenceOfParentArr[$j] & @CRLF)
141         ConsoleWrite("Parent MFT Ref SeqNo: " & $IndxMFTParentRefSeqNoArr[$j] & @CRLF)
142         ConsoleWrite("Flags: " & $IndxFileFlagsArr[$j] & @CRLF)
143         ConsoleWrite("File Create Time: " & $IndxCTimeArr[$j] & @CRLF)
144         ConsoleWrite("File Modified Time: " & $IndxATimeArr[$j] & @CRLF)
145         ConsoleWrite("MFT Entry modified Time: " & $IndxMTimeArr[$j] & @CRLF)
146         ConsoleWrite("File Last Access Time: " & $IndxRTimeArr[$j] & @CRLF)
147         ConsoleWrite("Allocated Size: " & $IndxAllocSizeArr[$j] & @CRLF)
148         ConsoleWrite("Real Size: " & $IndxRealSizeArr[$j] & @CRLF)
149         ConsoleWrite("NameSpace: " & $IndxNameSpaceArr[$j] & @CRLF)
150         ConsoleWrite("IndexFlags: " & $IndxIndexFlagsArr[$j] & @CRLF)
151         ConsoleWrite("SubNodeVCN: " & $IndxSubNodeVCNArr[$j] & @CRLF)
152         ConsoleWrite(@CRLF)
153     Next
154 ElseIf $AttributesArr[9][2] = "TRUE" Then ;And $ResidentIndx Then ; $INDEX_ROOT
155     ConsoleWrite("Directory listing for: " & $DirListPath & @CRLF & @CRLF)
156     For $j = 1 To Ubound($IndxFileNameArr)-1
157         ConsoleWrite("Entry number: " & $IndxEntryNumberArr[$j] & @CRLF)
158         ConsoleWrite("FileName: " & $IndxFileNameArr[$j] & @CRLF)
159         ConsoleWrite("MFT Ref: " & $IndxMFTReferenceArr[$j] & @CRLF)
160         ConsoleWrite("MFT Ref SeqNo: " & $IndxMFTRefSeqNoArr[$j] & @CRLF)
161         ConsoleWrite("Parent MFT Ref: " & $IndxMFTReferenceOfParentArr[$j] & @CRLF)
162         ConsoleWrite("Parent MFT Ref SeqNo: " & $IndxMFTParentRefSeqNoArr[$j] & @CRLF)
163         ConsoleWrite("Flags: " & $IndxFileFlagsArr[$j] & @CRLF)
164         ConsoleWrite("File Create Time: " & $IndxCTimeArr[$j] & @CRLF)
165         ConsoleWrite("File Modified Time: " & $IndxATimeArr[$j] & @CRLF)
166         ConsoleWrite("MFT Entry modified Time: " & $IndxMTimeArr[$j] & @CRLF)
167         ConsoleWrite("File Last Access Time: " & $IndxRTimeArr[$j] & @CRLF)
168         ConsoleWrite("Allocated Size: " & $IndxAllocSizeArr[$j] & @CRLF)
169         ConsoleWrite("Real Size: " & $IndxRealSizeArr[$j] & @CRLF)
170         ConsoleWrite("NameSpace: " & $IndxNameSpaceArr[$j] & @CRLF)
171         ConsoleWrite("IndexFlags: " & $IndxIndexFlagsArr[$j] & @CRLF)
172         ConsoleWrite("SubNodeVCN: " & $IndxSubNodeVCNArr[$j] & @CRLF)
173         ConsoleWrite(@CRLF)
174     Next
175 Else
176     ConsoleWrite("Error: There was no index found for the parent folder." & @CRLF)
177     Return SetError(1,0,0)
178 EndIf
179 ElseIf $DetailMode = 2 Then
180     If $AttributesArr[10][2] = "TRUE" Then; $INDEX_ALLOCATION
181         $HighestVal = _ArrayMax($IndxRealSizeArr,1)
182         If @error then
183             ConsoleWrite("Error: Unexpected error when resolving highest value in array: " & @error & @CRLF)
184             Exit
185         EndIf
186         $HighestValLength = StringLen($HighestVal)
187         $RealSizeStr = "RealSize"
188         If StringLen($RealSizeStr) < $HighestValLength Then $RealSizeStr = _AlignString($RealSizeStr,$HighestValLength)
189         ConsoleWrite("Directory listing for: " & $DirListPath & @CRLF & @CRLF)
190         ConsoleWrite("File Modified Time| File Type|" & $RealSizeStr & "| FileName" & @CRLF)
191         For $j = 1 To Ubound($IndxFileNameArr)-1
192             If StringInStr($IndxFileFlagsArr[$j],"directory") Then
193                 $FileType = "<DIR>"
194             Else
195                 $FileType = " "
196             EndIf
197             $AlignedSizeVal = _AlignString($IndxRealSizeArr[$j],$HighestValLength)
198             $TextOut = $IndxATimeArr[$j] & " | " & $FileType & " | " & $AlignedSizeVal & " | " & $IndxFileNameArr[$j] & @CRLF
199             ConsoleWrite($TextOut & @CRLF)
200         Next
201     ElseIf $AttributesArr[9][2] = "TRUE" Then ;And $ResidentIndx Then ; $INDEX_ROOT
202         $HighestVal = _ArrayMax($IndxRealSizeArr,1)
203         If @error then
204             ConsoleWrite("Error: Unexpected error when resolving highest value in array: " & @error & @CRLF)
205             Exit

```

```

205         EndIf
206         $HighestValLength = StringLen($HighestVal)
207         $RealSizeStr = "RealSize"
208         If StringLen($RealSizeStr) < $HighestValLength Then $RealSizeStr = _AlignString($RealSizeStr,$HighestValLength)
209         ConsoleWrite("Directory listing for: " & $DirListPath & @CRLF & @CRLF)
210         ConsoleWrite("          File Modified Time|   Type|"&$RealSizeStr"&"| FileName" & @CRLF)
211         For $j = 1 To Ubound($IdxFileNameArr)-1
212             If StringInStr($IdxFileFlagsArr[$j],"directory") Then
213                 $FileType = "<DIR>"
214             Else
215                 $FileType = "      "
216             EndIf
217             $AlignedSizeVal = _AlignString($IdxRealSizeArr[$j],$HighestValLength)
218             $TextOut = $IdxATimeArr[$j] & " | " & $FileType & " | " & $AlignedSizeVal & " | " & $IdxFileNameArr[$j]
219             ConsoleWrite($TextOut & @CRLF)
220         Next
221     Else
222         ; ConsoleWrite("Error: There was no index found for the parent folder." & @CRLF)
223         Return SetError(1,0,0)
224     EndIf
225 EndFunc
226
227 Func _ParseIndex($TestName)
228     If $AttributesArr[10][2] = "TRUE" Then; $INDEX_ALLOCATION
229         For $j = 1 To Ubound($IdxFileNameArr)-1
230             If $IdxFileNameArr[$j] = $TestName Then
231                 Return $IdxMFTReferenceArr[$j]
232             Else
233                 ;
234                 Return SetError(1,0,0)
235             EndIf
236         Next
237     ElseIf $AttributesArr[9][2] = "TRUE" Then ;And $ResidentIdx Then ; $INDEX_ROOT
238         For $j = 1 To Ubound($IdxFileNameArr)-1
239             If $IdxFileNameArr[$j] = $TestName Then
240                 Return $IdxMFTReferenceArr[$j]
241             Else
242                 ;
243                 Return SetError(1,0,0)
244             EndIf
245         Next
246     Else
247         ; ConsoleWrite("Error: No index found for: " & $TestName & @CRLF)
248         Return SetError(1,0,0)
249     EndIf
250 EndFunc
251
252 Func _GenDirArray($InPath)
253     Local $Reconstruct
254     Global $DirArray = StringSplit($InPath,"\")
255     For $i = 1 To $DirArray[0]-1
256         $Reconstruct &= $DirArray[$i]&"\"
257     Next
258     $Reconstruct = StringTrimRight($Reconstruct,1)
259     Return $Reconstruct
260 EndFunc
261
262 Func _ExtractSingleFile($MFTReferenceNumber)
263     Global $DataQ[1],$AttribX[1],$AttribXType[1],$AttribXCounter[1] ;clear array
264     $MFTRecord = _FindFileMFTRecord($MFTReferenceNumber)
265     If $MFTRecord = "" Then
266         ConsoleWrite("Target " & $MFTReferenceNumber & " not found" & @CRLF)
267         ;_DisplayInfo("Target " & $MFTReferenceNumber & " not found" & @CRLF)
268         Return SetError(1,0,0)
269     ElseIf StringMid($MFTRecord,3,8) <> $RecordSignature AND StringMid($MFTRecord,3,8) <> $RecordSignatureBad Then
270         ConsoleWrite("Found record is not valid:" & @CRLF)
271         ;_DisplayInfo("Found record is not valid:" & @CRLF)
272         ConsoleWrite(_HexEncode($MFTRecord) & @CRLF)
273         Return SetError(1,0,0)
274     EndIf
275     _DecodeMFTRecord($MFTRecord,1)
276     Return
277 EndFunc
278
279 Func _DecodeAttrList($TargetFile, $AttrList)

```

```

279 Local $offset, $length, $nBytes, $hFile, $LocalAttribID, $LocalName, $ALRecordLength, $ALNameLength, $ALNameOffset
280 If StringMid($AttrList, 17, 2) = "00" Then ;attribute List is in $AttrList
281     $offset = Dec(_SwapEndian(StringMid($AttrList, 41, 4)))
282     $List = StringMid($AttrList, $offset*2+1)
283 ;     $IsolatedAttributeList = $List

284 Else ;attribute List is found from data run in $AttrList
285     $size = Dec(_SwapEndian(StringMid($AttrList, $offset*2 + 97, 16)))
286     $offset = ($offset + Dec(_SwapEndian(StringMid($AttrList, $offset*2 + 65, 4))))*2
287     $DataRun = StringMid($AttrList, $offset+1, StringLen($AttrList)-$offset)
288 ;     ConsoleWrite("Attribute_List DataRun is " & $DataRun & @CRLF)
289     Global $RUN_VCN[1], $RUN_Clusters[1]
290     _ExtractDataRuns()
291     $tBuffer = DllStructCreate("byte[" & $BytesPerCluster & "]")
292     $hFile = _WinAPI_CreateFile("\\.\\" & $TargetDrive, 2, 6, 6)
293     If $hFile = 0 Then
294         ConsoleWrite("Error in function CreateFile when trying to locate Attribute List." & @CRLF)
295         ;_DisplayInfo("Error in function CreateFile when trying to locate Attribute List." & @CRLF)
296         _WinAPI_CloseHandle($hFile)
297         Return SetError(1,0,0)
298     EndIf
299     $List = ""
300     For $r = 1 To Ubound($RUN_VCN)-1
301         _WinAPI_SetFilePointerEx($hFile, $RUN_VCN[$r]*$BytesPerCluster, $FILE_BEGIN)
302         For $i = 1 To $RUN_Clusters[$r]
303             _WinAPI_ReadFile($hFile, DllStructGetPtr($tBuffer), $BytesPerCluster, $nBytes)
304             $List &= StringTrimLeft(DllStructGetData($tBuffer, 1),2)
305         Next
306     Next
307 ;     _DebugOut("****AttrList New:", $List)
308     _WinAPI_CloseHandle($hFile)
309     $List = StringMid($List, 1, $size*2)

310 EndIf
311 $IsolatedAttributeList = $list
312 $offset=0
313 $str=""
314 While StringLen($list) > $offset*2
315     $type=StringMid($List, ($offset*2)+1, 8)
316     $ALRecordLength = Dec(_SwapEndian(StringMid($List, $offset*2 + 9, 4)))
317     $ALNameLength = Dec(_SwapEndian(StringMid($List, $offset*2 + 13, 2)))
318     $ALNameOffset = Dec(_SwapEndian(StringMid($List, $offset*2 + 15, 2)))
319     $TestVCN = Dec(_SwapEndian(StringMid($List, $offset*2 + 17, 16)))
320     $ref=Dec(_SwapEndian(StringMid($List, $offset*2 + 33, 8)))
321     $LocalAttribID = "0x" & StringMid($List, $offset*2 + 49, 2) & StringMid($List, $offset*2 + 51, 2)
322     If $ALNameLength > 0 Then
323         $LocalName = StringMid($List, $offset*2 + 53, $ALNameLength*2*2)
324         $LocalName = _UnicodeHexToStr($LocalName)
325     Else
326         $LocalName = ""
327     EndIf
328     If $ref <> $TargetFile Then ;new attribute
329         If Not StringInStr($str, $ref) Then $str &= $ref & "- "
330     EndIf
331     If $type=$DATA Then
332         $DataInAttrlist=1
333         $IsolatedData=StringMid($List, ($offset*2)+1, $ALRecordLength*2)

334         If $TestVCN=0 Then $DataIsResident=1
335     EndIf
336     $offset += Dec(_SwapEndian(StringMid($List, $offset*2 + 9, 4)))
337 WEnd
338 If $str = "" Then
339     ConsoleWrite("No extra MFT records found" & @CRLF)
340     ;_DisplayInfo("No extra MFT records found" & @CRLF)
341 Else
342     $AttrQ = StringSplit(StringTrimRight($str,1), "-")
343 ;     ConsoleWrite("Decode of $ATTRIBUTE_LIST reveiled extra MFT Records to be examined = " & _ArrayToString($Attr
344 EndIf
345 EndFunc

346
347 Func _StripMftRecord($MFTEntry)
348     $UpdSeqArrOffset = Dec(_SwapEndian(StringMid($MFTEntry,11,4)))
349     $UpdSeqArrSize = Dec(_SwapEndian(StringMid($MFTEntry,15,4)))
350     $UpdSeqArr = StringMid($MFTEntry,3+($UpdSeqArrOffset*2),$UpdSeqArrSize*2*2)
351     $UpdSeqArrPart0 = StringMid($UpdSeqArr,1,4)

```

```

352     $UpdSeqArrPart1 = StringMid($UpdSeqArr,5,4)
353     $UpdSeqArrPart2 = StringMid($UpdSeqArr,9,4)
354     $RecordEnd1 = StringMid($MFTEntry,1023,4)
355     $RecordEnd2 = StringMid($MFTEntry,2047,4)
356     If $UpdSeqArrPart0 <> $RecordEnd1 OR $UpdSeqArrPart0 <> $RecordEnd2 Then
357         ConsoleWrite("Error the $MFT record is corrupt" & @CRLF)
358         ;_DisplayInfo("Error the $MFT record is corrupt" & @CRLF)
359         Return SetError(1,0,0)
360     Else
361         $MFTEntry = StringMid($MFTEntry,1,1022) & $UpdSeqArrPart1 & StringMid($MFTEntry,1027,1020) & $UpdSeqArrPart2
362     EndIf
363     $RecordSize = Dec(_SwapEndian(StringMid($MFTEntry,51,8)),2)
364     $HeaderSize = Dec(_SwapEndian(StringMid($MFTEntry,43,4)),2)
365     $MFTEntry = StringMid($MFTEntry,$HeaderSize*2+3,($RecordSize-$HeaderSize-8)*2) ;strip "0x..." and "FFFFFFF."
366     Return $MFTEntry
367 EndFunc
368
369 Func _DecodeDataQEntry($attr) ;processes data attribute
370     $NonResidentFlag = StringMid($attr,17,2)
371     $NameLength = Dec(StringMid($attr,19,2))
372     $NameOffset = Dec(_SwapEndian(StringMid($attr,21,4)))
373     If $NameLength > 0 Then ;must be ADS
374         $ADS_Name = _UnicodeHexToStr(StringMid($attr,$NameOffset*2 + 1,$NameLength*4))
375         $ADS_Name = $FN_FileName & "[ADS_" & $ADS_Name & "]"
376     Else
377         $ADS_Name = $FN_FileName ;need to preserve $FN_FileName
378     EndIf
379     $Flags = StringMid($attr,25,4)
380     If BitAND($Flags,"0100") Then $IsCompressed = 1
381     If BitAND($Flags,"0080") Then $IsSparse = 1
382     If $NonResidentFlag = '01' Then
383         $DATA_Clusters = Dec(_SwapEndian(StringMid($attr,49,16)),2) - Dec(_SwapEndian(StringMid($attr,33,16)),2) + 1
384
385         $DATA_RealSize = Dec(_SwapEndian(StringMid($attr,97,16)),2)
386         $DATA_InitSize = Dec(_SwapEndian(StringMid($attr,113,16)),2)
387         $Offset = Dec(_SwapEndian(StringMid($attr,65,4)))
388         $DataRun = StringMid($attr,$Offset*2+1,(StringLen($attr)-$Offset)*2)
389     ElseIf $NonResidentFlag = '00' Then
390         $DATA_LengthOfAttribute = Dec(_SwapEndian(StringMid($attr,33,8)),2)
391         $Offset = Dec(_SwapEndian(StringMid($attr,41,4)))
392         $DataRun = StringMid($attr,$Offset*2+1,$DATA_LengthOfAttribute*2)
393     EndIf
394 EndFunc
395
396 Func _DecodeMFTRecord($MFTEntry,$MFTMode)
397 Global $IdxEntryNumberArr[1],$IdxMFTReferenceArr[1],$IdxIndexFlagsArr[1],$IdxMFTReferenceOfParentArr[1],$IdxCTimeArr[1]
398 Local $MFTEntryOrig,$FN_Number,$DATA_Number,$SI_Number,$ATTRIBLIST_Number,$OBJID_Number,$SECURITY_Number,$VOLNAME_Number,$V
399 Local $INDEX_ROOT_ON="FALSE",$INDEX_ALLOCATION_ON="FALSE"
400 Global $IRArr[12][2],$IdxArr[20][2]
401 _SetArrays()
402 $HEADER_RecordRealSize = ""
403 $HEADER_MFTRecordNumber = ""
404 $UpdSeqArrOffset = Dec(_SwapEndian(StringMid($MFTEntry,11,4)))
405 $UpdSeqArrSize = Dec(_SwapEndian(StringMid($MFTEntry,15,4)))
406 $UpdSeqArr = StringMid($MFTEntry,3+($UpdSeqArrOffset*2),$UpdSeqArrSize*2*2)
407 $UpdSeqArrPart0 = StringMid($UpdSeqArr,1,4)
408 $UpdSeqArrPart1 = StringMid($UpdSeqArr,5,4)
409 $UpdSeqArrPart2 = StringMid($UpdSeqArr,9,4)
410 $RecordEnd1 = StringMid($MFTEntry,1023,4)
411 $RecordEnd2 = StringMid($MFTEntry,2047,4)
412 If $UpdSeqArrPart0 <> $RecordEnd1 OR $UpdSeqArrPart0 <> $RecordEnd2 Then
413     ConsoleWrite("Error: the $MFT record is corrupt" & @CRLF)
414     ;_DisplayInfo("Error: the $MFT record is corrupt" & @CRLF)
415     Return SetError(1,0,0)
416 Else
417     $MFTEntry = StringMid($MFTEntry,1,1022) & $UpdSeqArrPart1 & StringMid($MFTEntry,1027,1020) & $UpdSeqArrPart2
418 EndIf
419 $HEADER_RecordRealSize = Dec(_SwapEndian(StringMid($MFTEntry,51,8)),2)
420 If $UpdSeqArrOffset = 48 Then
421     $HEADER_MFTRecordNumber = Dec(_SwapEndian(StringMid($MFTEntry,91,8)),2)
422 Else
423     $HEADER_MFTRecordNumber = "NT style"
424 EndIf
425 $AttributeOffset = (Dec(StringMid($MFTEntry,43,2))*2)+3
426

```

```

426 while 1
427     $AttributeType = StringMid($MFTEntry,$AttributeOffset,8)
428     $AttributeSize = StringMid($MFTEntry,$AttributeOffset+8,8)
429     $AttributeSize = Dec(_SwapEndian($AttributeSize),2)
430     Select
431         Case $AttributeType = $STANDARD_INFORMATION
432             ;
433             $STANDARD_INFORMATION_ON = "TRUE"
434             $SI_Number += 1
435
436             If $MFTMode = 1 Then
437                 _ArrayAdd($AttribX, StringMid($MFTEntry,$AttributeOffset,$AttributeSize*2))
438                 _ArrayAdd($AttribXType, $AttributeType)
439                 _ArrayAdd($AttribXCounter, $SI_Number)
440             EndIf
441         Case $AttributeType = $ATTRIBUTE_LIST
442             ;
443             $ATTRIBUTE_LIST_ON = "TRUE"
444             $ATTRIBLIST_Number += 1
445             If $MFTMode = 1 Then
446                 _ArrayAdd($AttribX, StringMid($MFTEntry,$AttributeOffset,$AttributeSize*2))
447                 _ArrayAdd($AttribXType, $AttributeType)
448                 _ArrayAdd($AttribXCounter, $ATTRIBLIST_Number)
449             EndIf
450             $MFTEntryOrig = $MFTEntry
451             $AttrList = StringMid($MFTEntry,$AttributeOffset,$AttributeSize*2)
452             _DecodeAttrList($HEADER_MFTRecordNumber, $AttrList) ;produces $AttrQ - extra record lis
453             $str = ""
454             For $i = 1 To $AttrQ[0]
455                 $record = _FindFileMFTRecord($AttrQ[$i])
456                 $str &= _StripMftRecord($record) ;no header or end marker
457             Next
458             $str &= "FFFFFFFF" ;add end marker
459             $MFTEntry = StringMid($MFTEntry,1,($HEADER_RecordRealSize-8)*2+2) & $str ;strip "FFFFFFFF..."
460         Case $AttributeType = $FILE_NAME
461             ;
462             $FILE_NAME_ON = "TRUE"
463             $FN_Number += 1
464             If $MFTMode = 1 Then
465                 _ArrayAdd($AttribX, StringMid($MFTEntry,$AttributeOffset,$AttributeSize*2))
466                 _ArrayAdd($AttribXType, $AttributeType)
467                 _ArrayAdd($AttribXCounter, $FN_Number)
468             EndIf
469             $attr = StringMid($MFTEntry,$AttributeOffset,$AttributeSize*2)
470             $NameSpace = StringMid($attr,179,2)
471             Select
472                 Case $NameSpace = "00" ;POSIX
473                     $NameQ[2] = $attr
474                 Case $NameSpace = "01" ;WIN32
475                     $NameQ[4] = $attr
476                 Case $NameSpace = "02" ;DOS
477                     $NameQ[1] = $attr
478                 Case $NameSpace = "03" ;DOS+WIN32
479                     $NameQ[3] = $attr
480             EndSelect
481         Case $AttributeType = $OBJECT_ID
482             ;
483             $OBJECT_ID_ON = "TRUE"
484             $OBJID_Number += 1
485             If $MFTMode = 1 Then
486                 _ArrayAdd($AttribX, StringMid($MFTEntry,$AttributeOffset,$AttributeSize*2))
487                 _ArrayAdd($AttribXType, $AttributeType)
488                 _ArrayAdd($AttribXCounter, $OBJID_Number)
489             EndIf
490         Case $AttributeType = $SECURITY_DESCRIPTOR
491             ;
492             $SECURITY_DESCRIPTOR_ON = "TRUE"
493             $SECURITY_Number += 1
494             If $MFTMode = 1 Then
495                 _ArrayAdd($AttribX, StringMid($MFTEntry,$AttributeOffset,$AttributeSize*2))
496                 _ArrayAdd($AttribXType, $AttributeType)
497                 _ArrayAdd($AttribXCounter, $SECURITY_Number)
498             EndIf
499         Case $AttributeType = $VOLUME_NAME
500             ;
501             $VOLUME_NAME_ON = "TRUE"
502             $VOLNAME_Number += 1
503             If $MFTMode = 1 Then
504                 _ArrayAdd($AttribX, StringMid($MFTEntry,$AttributeOffset,$AttributeSize*2))
505                 _ArrayAdd($AttribXType, $AttributeType)
506                 _ArrayAdd($AttribXCounter, $VOLNAME_Number)
507             EndIf

```



```

477         _ArrayAdd($AttributeList, $VOLUME_NAME)
478     EndIf
479
480     Case $AttributeType = $VOLUME_INFORMATION
481         $VOLUME_INFORMATION_ON = "TRUE"
482         $VOLINFO_Number += 1
483         If $MFTMode = 1 Then
484             _ArrayAdd($AttribX, StringMid($MFTEntry,$AttributeOffset,$AttributeSize*2))
485             _ArrayAdd($AttribXType, $AttributeType)
486             _ArrayAdd($AttribXCounter, $VOLINFO_Number)
487         EndIf
488
489     Case $AttributeType = $DATA
490         $DATA_ON = "TRUE"
491         $DATA_Number += 1
492         _ArrayAdd($DataQ, StringMid($MFTEntry,$AttributeOffset,$AttributeSize*2))
493
494     Case $AttributeType = $INDEX_ROOT
495         $INDEX_ROOT_ON = "TRUE"
496         $INDEXROOT_Number += 1
497         If $MFTMode = 1 Then
498             _ArrayAdd($AttribX, StringMid($MFTEntry,$AttributeOffset,$AttributeSize*2))
499             _ArrayAdd($AttribXType, $AttributeType)
500             _ArrayAdd($AttribXCounter, $INDEXROOT_Number)
501         EndIf
502         ReDim $IRArr[12][$INDEXROOT_Number+1]
503         $CoreIndexRoot = _GetAttributeEntry(StringMid($MFTEntry,$AttributeOffset,$AttributeSize*2))
504         $CoreIndexRootChunk = $CoreIndexRoot[0]
505         $CoreIndexRootName = $CoreIndexRoot[1]
506         If $CoreIndexRootName = "$I30" Then _Get_IndexRoot($CoreIndexRootChunk,$INDEXROOT_Number,$CoreIndex
507
508     Case $AttributeType = $INDEX_ALLOCATION
509         $INDEX_ALLOCATION_ON = "TRUE"
510         $INDEXALLOC_Number += 1
511         If $MFTMode = 1 Then
512             _ArrayAdd($AttribX, StringMid($MFTEntry,$AttributeOffset,$AttributeSize*2))
513             _ArrayAdd($AttribXType, $AttributeType)
514             _ArrayAdd($AttribXCounter, $INDEXALLOC_Number)
515         EndIf
516
517         $CoreIndexAllocation = _GetAttributeEntry(StringMid($MFTEntry,$AttributeOffset,$AttributeSize*2))
518         $CoreIndexAllocationChunk = $CoreIndexAllocation[0]
519         $CoreIndexAllocationName = $CoreIndexAllocation[1]
520         _ArrayAdd($HexDumpIndexRecord,$CoreIndexAllocationChunk)
521         If $CoreIndexAllocationName = "$I30" Then _Get_IndexAllocation($CoreIndexAllocationChunk,$INDEXALLO
522
523     Case $AttributeType = $BITMAP
524         $BITMAP_ON = "TRUE"
525         $BITMAP_Number += 1
526         If $MFTMode = 1 Then
527             _ArrayAdd($AttribX, StringMid($MFTEntry,$AttributeOffset,$AttributeSize*2))
528             _ArrayAdd($AttribXType, $AttributeType)
529             _ArrayAdd($AttribXCounter, $BITMAP_Number)
530         EndIf
531
532     Case $AttributeType = $REPARSE_POINT
533         $REPARSE_POINT_ON = "TRUE"
534         $REPARSEPOINT_Number += 1
535         If $MFTMode = 1 Then
536             _ArrayAdd($AttribX, StringMid($MFTEntry,$AttributeOffset,$AttributeSize*2))
537             _ArrayAdd($AttribXType, $AttributeType)
538             _ArrayAdd($AttribXCounter, $REPARSEPOINT_Number)
539         EndIf
540
541     Case $AttributeType = $EA_INFORMATION
542         $EA_INFORMATION_ON = "TRUE"
543         $EAINFO_Number += 1
544         If $MFTMode = 1 Then
545             _ArrayAdd($AttribX, StringMid($MFTEntry,$AttributeOffset,$AttributeSize*2))
546             _ArrayAdd($AttribXType, $AttributeType)
547             _ArrayAdd($AttribXCounter, $EAINFO_Number)
548         EndIf
549
550     Case $AttributeType = $EA
551         $EA_ON = "TRUE"
552         $EA_Number += 1
553         If $MFTMode = 1 Then
554             _ArrayAdd($AttribX, StringMid($MFTEntry,$AttributeOffset,$AttributeSize*2))
555             _ArrayAdd($AttribXType, $AttributeType)
556             _ArrayAdd($AttribXCounter, $EA_Number)
557         EndIf
558
559     Case $AttributeType = $PROPERTY_SET
560         $PROPERTY_SET_ON = "TRUE"
561         $PROPERTYSET_Number += 1

```



```

574     If $MFTMode = 1 Then
575         _ArrayAdd($AttribX, StringMid($MFTEntry,$AttributeOffset,$AttributeSize*2))
576         _ArrayAdd($AttribXType, $AttributeType)
577         _ArrayAdd($AttribXCounter, $PROPERTYSET_Number)
578     EndIf
579     Case $AttributeType = $LOGGED_UTILITY_STREAM
580 ;         $LOGGED_UTILITY_STREAM_ON = "TRUE"
581         $LOGGEDUTILSTREAM_Number += 1
582         If $MFTMode = 1 Then
583             _ArrayAdd($AttribX, StringMid($MFTEntry,$AttributeOffset,$AttributeSize*2))
584             _ArrayAdd($AttribXType, $AttributeType)
585             _ArrayAdd($AttribXCounter, $LOGGEDUTILSTREAM_Number)
586         EndIf
587         Case $AttributeType = $ATTRIBUTE_END_MARKER
588             ExitLoop
589     EndSelect
590     $AttributeOffset += $AttributeSize*2
591 WEnd
592 $AttributesArr[9][2] = $INDEX_ROOT_ON
593 $AttributesArr[10][2] = $INDEX_ALLOCATION_ON
594 EndFunc
595
596 Func _ExtractDataRuns()
597     $r=UBound($RUN_Clusters)
598     $i=1
599     $RUN_VCN[0] = 0
600     $BaseVCN = $RUN_VCN[0]
601     If $DataRun = "" Then $DataRun = "00"
602     Do
603         $RunListID = StringMid($DataRun,$i,2)
604         If $RunListID = "00" Then ExitLoop
605         $i += 2
606         $RunListClustersLength = Dec(StringMid($RunListID,2,1))
607         $RunListVCNLength = Dec(StringMid($RunListID,1,1))
608         $RunListClusters = Dec(_SwapEndian(StringMid($DataRun,$i,$RunListClustersLength*2)),2)
609         $i += $RunListClustersLength*2
610         $RunListVCN = _SwapEndian(StringMid($DataRun, $i, $RunListVCNLength*2))
611         ;next line handles positive or negative move
612         $BaseVCN += Dec($RunListVCN,2)-((($r>1) And (Dec(StringMid($RunListVCN,1,1))>7))*Dec(StringMid("100000000000",1,1)))
613         If $RunListVCN <> "" Then
614             $RunListVCN = $BaseVCN
615         Else
616             $RunListVCN = 0 ;$RUN_VCN[$r-1] ;0
617         EndIf
618         If (($RunListVCN=0) And ($RunListClusters>16) And (Mod($RunListClusters,16)>0)) Then
619             ;may be sparse section at end of Compression Signature
620             _ArrayAdd($RUN_Clusters,Mod($RunListClusters,16))
621             _ArrayAdd($RUN_VCN,$RunListVCN)
622             $RunListClusters -= Mod($RunListClusters,16)
623             $r += 1
624         ElseIf (($RunListClusters>16) And (Mod($RunListClusters,16)>0)) Then
625             ;may be compressed data section at start of Compression Signature
626             _ArrayAdd($RUN_Clusters,$RunListClusters-Mod($RunListClusters,16))
627             _ArrayAdd($RUN_VCN,$RunListVCN)
628             $RunListVCN += $RUN_Clusters[$r]
629             $RunListClusters = Mod($RunListClusters,16)
630             $r += 1
631         EndIf
632         ;just normal or sparse data
633         _ArrayAdd($RUN_Clusters,$RunListClusters)
634
635         _ArrayAdd($RUN_VCN,$RunListVCN)
636         $r += 1
637         $i += $RunListVCNLength*2
638     Until $i > StringLen($DataRun)
639 EndFunc
640
641 Func _FindFileMFTRecord($TargetFile)
642     Local $nBytes, $TmpOffset, $Counter, $Counter2, $RecordJumper, $TargetFileDec, $RecordsTooMuch, $RetVal[2]
643     $tBuffer = DllStructCreate("byte[" & $MFT_Record_Size & "]")
644     $hFile = _WinAPI_CreateFile("\\.\\" & $TargetDrive, 2, 6, 6)
645     If $hFile = 0 Then
646         ConsoleWrite("Error in function CreateFile: " & _WinAPI_GetLastError() & @CRLF)
647         : DisplavInfo("Error in function CreateFile: " & WinAPI GetLastError() & @CRLF)

```

```

647     _WinAPI_CloseHandle($hFile)
648     Return SetError(1,0,0)
649 EndIf
650 $TargetFile = _DecToLittleEndian($TargetFile)
651 $TargetFileDec = Dec(_SwapEndian($TargetFile),2)
652 For $i = 1 To UBound($MFT_RUN_Clusters)-1
653     $CurrentClusters = $MFT_RUN_Clusters[$i]
654     $RecordsInCurrentRun = ($CurrentClusters*$SectorsPerCluster)/2
655     $Counter+=$RecordsInCurrentRun
656     If $Counter>$TargetFileDec Then
657         ExitLoop
658     EndIf
659 Next
660 $TryAt = $Counter-$RecordsInCurrentRun
661 $TryAtArrIndex = $i
662 $RecordsPerCluster = $SectorsPerCluster/2
663 Do
664     $RecordJumper+=$RecordsPerCluster
665     $Counter2+=1
666     $Final = $TryAt+$RecordJumper
667 Until $Final>=$TargetFileDec
668 $RecordsTooMuch = $Final-$TargetFileDec
669 _WinAPI_SetFilePointerEx($hFile, $ImageOffset+$MFT_RUN_VCN[$i]*$BytesPerCluster+($Counter2*$BytesPerCluster)-($RecordsTooMuch), $FILE_BEGIN, 0)
670 _WinAPI_ReadFile($hFile, DllStructGetPtr($tBuffer), $MFT_Record_Size, $nBytes)
671 $record = DllStructGetData($tBuffer, 1)
672 If StringMid($record,91,8) = $TargetFile Then
673     $TmpOffset = DllCall('kernel32.dll', 'int', 'SetFilePointerEx', 'ptr', $hFile, 'int64', 0, 'int64*', 0, 'dw')
674 ; ConsoleWrite("Record number: " & Dec(_SwapEndian($TargetFile),2) & " found at disk offset: " & $TmpOffset[3] & @CRLF)
675 ; _DisplayInfo("Record number: " & Dec(_SwapEndian($TargetFile),2) & " found at disk offset: " & $TmpOffset[3] & @CRLF)
676 _WinAPI_CloseHandle($hFile)
677 ; $RetVal[0] = $TmpOffset[3]-1024
678 ; $RetVal[1] = $record
679 ; Return $RetVal
680 Return $record
681 Else
682     _WinAPI_CloseHandle($hFile)
683     Return ""
684 EndIf
685 EndFunc
686
687 Func _FindMFT($TargetFile)
688     Local $nBytes;, $MFT_Record_Size=1024
689     $tBuffer = DllStructCreate("byte[" & $MFT_Record_Size & "]")
690     $hFile = _WinAPI_CreateFile("\\.\\" & $TargetDrive, 2, 2, 7)
691     If $hFile = 0 Then
692         ConsoleWrite("Error in function CreateFile when trying to locate MFT: " & _WinAPI_GetLastErrorMessage() & @CRLF)
693         ; _DisplayInfo("Error in function CreateFile when trying to locate MFT: " & _WinAPI_GetLastErrorMessage() & @CRLF)
694         Return SetError(1,0,0)
695     EndIf
696 ; ConsoleWrite("$MFT_Offset: " & $MFT_Offset & @CRLF)
697 _WinAPI_SetFilePointerEx($hFile, $ImageOffset+$MFT_Offset, $FILE_BEGIN)
698 _WinAPI_ReadFile($hFile, DllStructGetPtr($tBuffer), $MFT_Record_Size, $nBytes)
699 _WinAPI_CloseHandle($hFile)
700 $record = DllStructGetData($tBuffer, 1)
701 If NOT StringMid($record,1,8) = '46494C45' Then
702     ConsoleWrite("MFT record signature not found. "& @CRLF)
703     ; _DisplayInfo("MFT record signature not found. "& @CRLF)
704     Return ""
705 EndIf
706 If StringMid($record,47,4) = "0100" AND Dec(_SwapEndian(StringMid($record,91,8))) = $TargetFile Then
707 ; ConsoleWrite("MFT record found" & @CRLF)
708 Return $record ;returns record for MFT
709 EndIf
710 ConsoleWrite("MFT record not found" & @CRLF)
711 ; _DisplayInfo("MFT record not found" & @CRLF)
712 Return ""
713 EndFunc
714
715 Func _DecToLittleEndian($DecimalInput)
716     Return _SwapEndian(Hex($DecimalInput,8))
717 EndFunc
718
719 Func _SwapEndian($iHex)
720     Return StringMid(Binary(Dec($iHex,2)),3, StringLen($iHex))

```

```

721 EndFunc
722
723 Func _UnicodeHexToStr($FileName)
724     $str = ""
725     For $i = 1 To StringLen($FileName) Step 4
726         $str &= ChrW(Dec(_SwapEndian(StringMid($FileName, $i, 4))))
727     Next
728     Return $str
729 EndFunc
730
731 Func _DebugOut($text, $var)
732     ConsoleWrite("Debug output for " & $text & @CRLF)
733     For $i=1 To StringLen($var) Step 32
734
735         $str=""
736         For $n=0 To 15
737             $str &= StringMid($var, $i+$n*2, 2) & " "
738             if $n=7 then $str &= "- "
739         Next
740         ConsoleWrite($str & @CRLF)
741     Next
742 EndFunc
743
744 Func _ReadBootSector($TargetDrive)
745     Local $nbytes
746     $tBuffer=DllStructCreate("byte[512]")
747     $hFile = _WinAPI_CreateFile("\\.\\" & $TargetDrive,2,2,7)
748     If $hFile = 0 then
749         ConsoleWrite("Error in function CreateFile: " & _WinAPI_GetLastError() & " for: " & "\\.\\" & $TargetDrive)
750         ;_DisplayInfo("Error in function CreateFile: " & _WinAPI_GetLastError() & " for: " & "\\.\\" & $TargetDrive)
751         Return SetError(1,0,0)
752     EndIf
753     _WinAPI_SetFilePointerEx($hFile, $ImageOffset, $FILE_BEGIN)
754     $read = _WinAPI_ReadFile($hFile, DllStructGetPtr($tBuffer), 512, $nBytes)
755     If $read = 0 then
756         ConsoleWrite("Error in function ReadFile: " & _WinAPI_GetLastError() & " for: " & "\\.\\" & $TargetDrive)
757         ;_DisplayInfo("Error in function ReadFile: " & _WinAPI_GetLastError() & " for: " & "\\.\\" & $TargetDrive)
758         Return
759     EndIf
760     _WinAPI_CloseHandle($hFile)
761     ; Good starting point from KaFu & trancexx at the AutoIt forum
762     $tBootSectorSections = DllStructCreate("align 1;" & _
763         "byte Jump[3];" & _
764         "char SystemName[8];" & _
765         "ushort BytesPerSector;" & _
766         "ubyte SectorsPerCluster;" & _
767         "ushort ReservedSectors;" & _
768         "ubyte[3];" & _
769         "ushort;" & _
770         "ubyte MediaDescriptor;" & _
771         "ushort;" & _
772         "ushort SectorsPerTrack;" & _
773         "ushort NumberOfHeads;" & _
774         "dword HiddenSectors;" & _
775         "dword;" & _
776         "dword;" & _
777         "int64 TotalSectors;" & _
778         "int64 LogicalClusterNumberfortheFileMFT;" & _
779         "int64 LogicalClusterNumberfortheFileMFTMirr;" & _
780         "dword ClustersPerFileRecordSegment;" & _
781         "dword ClustersPerIndexBlock;" & _
782         "int64 NTFSVolumeSerialNumber;" & _
783         "dword Checksum", DllStructGetPtr($tBuffer))
784
785     $BytesPerSector = DllStructGetData($tBootSectorSections, "BytesPerSector")
786     $SectorsPerCluster = DllStructGetData($tBootSectorSections, "SectorsPerCluster")
787     $BytesPerCluster = $BytesPerSector * $SectorsPerCluster
788     $ClustersPerFileRecordSegment = DllStructGetData($tBootSectorSections, "ClustersPerFileRecordSegment")
789     $LogicalClusterNumberfortheFileMFT = DllStructGetData($tBootSectorSections, "LogicalClusterNumberfortheFileMFT")
790     $MFT_Offset = $BytesPerCluster * $LogicalClusterNumberfortheFileMFT
791     If $ClustersPerFileRecordSegment > 127 Then
792         $MFT_Record_Size = 2 ^ (256 - $ClustersPerFileRecordSegment)
793     Else
794         $MFT_Record_Size = $BytesPerCluster * $ClustersPerFileRecordSegment

```

```

794     EndIf
795 EndFunc
796
797 Func _HexEncode($bInput)
798     Local $tInput = DllStructCreate("byte[" & BinaryLen($bInput) & "]")
799     DllStructSetData($tInput, 1, $bInput)
800     Local $a_iCall = DllCall("crypt32.dll", "int", "CryptBinaryToString", _
801         "ptr", DllStructGetPtr($tInput), _
802         "dword", DllStructGetSize($tInput), _
803         "dword", 11, _
804         "ptr", 0, _
805         "dword*", 0)
806
807     If @error Or Not $a_iCall[0] Then
808         Return SetError(1, 0, "")
809     EndIf
810
811     Local $iSize = $a_iCall[5]
812     Local $tOut = DllStructCreate("char[" & $iSize & "]")
813
814     $a_iCall = DllCall("crypt32.dll", "int", "CryptBinaryToString", _
815         "ptr", DllStructGetPtr($tInput), _
816         "dword", DllStructGetSize($tInput), _
817         "dword", 11, _
818         "ptr", DllStructGetPtr($tOut), _
819         "dword*", $iSize)
820
821     If @error Or Not $a_iCall[0] Then
822         Return SetError(2, 0, "")
823     EndIf
824
825     Return SetError(0, 0, DllStructGetData($tOut, 1))
826
827 EndFunc ;==>_HexEncode
828
829 Func _File_Attributes($FAInput)
830     Local $FAOutput = ""
831     If BitAND($FAInput, 0x0001) Then $FAOutput &= 'read_only+'
832     If BitAND($FAInput, 0x0002) Then $FAOutput &= 'hidden+'
833     If BitAND($FAInput, 0x0004) Then $FAOutput &= 'system+'
834
835     If BitAND($FAInput, 0x0010) Then $FAOutput &= 'directory+'
836     If BitAND($FAInput, 0x0020) Then $FAOutput &= 'archive+'
837     If BitAND($FAInput, 0x0040) Then $FAOutput &= 'device+'
838     If BitAND($FAInput, 0x0080) Then $FAOutput &= 'normal+'
839     If BitAND($FAInput, 0x0100) Then $FAOutput &= 'temporary+'
840     If BitAND($FAInput, 0x0200) Then $FAOutput &= 'sparse_file+'
841     If BitAND($FAInput, 0x0400) Then $FAOutput &= 'reparse_point+'
842     If BitAND($FAInput, 0x0800) Then $FAOutput &= 'compressed+'
843     If BitAND($FAInput, 0x1000) Then $FAOutput &= 'offline+'
844     If BitAND($FAInput, 0x2000) Then $FAOutput &= 'not_indexed+'
845     If BitAND($FAInput, 0x4000) Then $FAOutput &= 'encrypted+'
846     If BitAND($FAInput, 0x8000) Then $FAOutput &= 'integrity_stream+'
847     If BitAND($FAInput, 0x10000) Then $FAOutput &= 'virtual+'
848     If BitAND($FAInput, 0x20000) Then $FAOutput &= 'no_scrub_data+'
849     If BitAND($FAInput, 0x10000000) Then $FAOutput &= 'directory+'
850     If BitAND($FAInput, 0x20000000) Then $FAOutput &= 'index_view+'
851     $FAOutput = StringTrimRight($FAOutput, 1)
852     Return $FAOutput
853 EndFunc
854
855 Func _End($begin)
856     Local $timerdiff = TimerDiff($begin)
857     $timerdiff = Round(($timerdiff / 1000), 2)
858     ConsoleWrite("Job took " & $timerdiff & " seconds" & @CRLF)
859     ;_DisplayInfo("Job took " & $timerdiff & " seconds" & @CRLF)
860     Exit
861 EndFunc
862
863 Func _ExtractFile($record)
864     $cBuffer = DllStructCreate("byte[" & $BytesPerCluster * 16 & "]")
865     $zflag = 0
866     $hFile = _WinAPI_CreateFile($AttributeOutFileName, 3, 6, 7)
867     If $hFile Then
868         Select

```

```

868         Case UBound($RUN_VCN) = 1                ;no data, do nothing
869         Case UBound($RUN_VCN) = 2                ;may be normal or sparse
870             If $RUN_VCN[1] = 0 And $IsSparse Then    ;sparse
871                 $FileSize = _DoSparse(1, $hFile, $DATA_InitSize)
872             Else                                     ;normal
873                 $FileSize = _DoNormal(1, $hFile, $cBuffer, $DATA_InitSize)
874             EndIf
875         Case Else                                    ;may be compressed
876             _DoCompressed($hFile, $cBuffer, $record)
877     EndSelect
878     If $DATA_RealSize > $DATA_InitSize Then
879         $FileSize = _WriteZeros($hfile, $DATA_RealSize - $DATA_InitSize)
880     EndIf
881     _WinAPI_CloseHandle($hFile)
882     Return
883 Else
884     ConsoleWrite("Error creating output file: " & _WinAPI_GetLastErrorMessage() & @CRLF)
885     ;_DisplayInfo("Error creating output file: " & _WinAPI_GetLastErrorMessage() & @CRLF)
886 EndIf
887 EndFunc
888
889 Func _WriteZeros($hfile, $count)
890     Local $nBytes
891     If Not IsDllStruct($sBuffer) Then _CreateSparseBuffer()
892     While $count > $BytesPerCluster * 16
893         _WinAPI_WriteFile($hFile, DllStructGetPtr($sBuffer), $BytesPerCluster * 16, $nBytes)
894         $count -= $BytesPerCluster * 16
895         $ProgressSize = $DATA_RealSize - $count
896     WEnd
897     If $count <> 0 Then _WinAPI_WriteFile($hFile, DllStructGetPtr($sBuffer), $count, $nBytes)
898     $ProgressSize = $DATA_RealSize
899     Return 0
900 EndFunc
901
902 Func _DoCompressed($hFile, $cBuffer, $record)
903     Local $nBytes
904     $r=1
905     $FileSize = $DATA_InitSize
906     $ProgressSize = $FileSize
907     Do
908         _WinAPI_SetFilePointerEx($hDisk, $ImageOffset+$RUN_VCN[$r]*$BytesPerCluster, $FILE_BEGIN)
909         $i = $RUN_Clusters[$r]
910         If (($RUN_VCN[$r+1]=0) And ($i+$RUN_Clusters[$r+1]=16) And $IsCompressed) Then
911             _WinAPI_ReadFile($hDisk, DllStructGetPtr($cBuffer), $BytesPerCluster * $i, $nBytes)
912             $Decompressed = _LZNTDecompress($cBuffer, $BytesPerCluster * $i)
913             If IsString($Decompressed) Then
914                 If $r = 1 Then
915                     _DebugOut("Decompression error for " & $ADS_Name, $record)
916                 Else
917                     _DebugOut("Decompression error (partial write) for " & $ADS_Name, $record)
918                 EndIf
919                 Return
920             Else ;$Decompressed is an array
921                 Local $dBuffer = DllStructCreate("byte[" & $Decompressed[1] & "]")
922                 DllStructSetData($dBuffer, 1, $Decompressed[0])
923             EndIf
924             If $FileSize > $Decompressed[1] Then
925                 _WinAPI_WriteFile($hFile, DllStructGetPtr($dBuffer), $Decompressed[1], $nBytes)
926                 $FileSize -= $Decompressed[1]
927                 $ProgressSize = $FileSize
928             Else
929                 _WinAPI_WriteFile($hFile, DllStructGetPtr($dBuffer), $FileSize, $nBytes)
930             EndIf
931             $r += 1
932         ElseIf $RUN_VCN[$r]=0 Then
933             $FileSize = _DoSparse($r, $hFile, $FileSize)
934             $ProgressSize = 0
935         Else
936             $FileSize = _DoNormal($r, $hFile, $cBuffer, $FileSize)
937             $ProgressSize = 0
938         EndIf
939         $r += 1
940     Until $r > UBound($RUN_VCN)-2

```

```

941     If $r = UBound($RUN_VCN)-1 Then
942         If $RUN_VCN[$r]=0 Then
943             $FileSize = _DoSparse($r, $hFile, $FileSize)
944             $ProgressSize = 0
945         Else
946             $FileSize = _DoNormal($r, $hFile, $cBuffer, $FileSize)
947             $ProgressSize = 0
948         EndIf
949     EndIf
950 EndFunc
951
952 Func _DoNormal($r, $hFile, $cBuffer, $FileSize)
953     Local $nBytes
954     _WinAPI_SetFilePointerEx($hDisk, $ImageOffset+$RUN_VCN[$r]*$BytesPerCluster, $FILE_BEGIN)
955     $i = $RUN_Clusters[$r]
956     While $i > 16 And $FileSize > $BytesPerCluster * 16
957         _WinAPI_ReadFile($hDisk, DllStructGetPtr($cBuffer), $BytesPerCluster * 16, $nBytes)
958         _WinAPI_WriteFile($hFile, DllStructGetPtr($cBuffer), $BytesPerCluster * 16, $nBytes)
959         $i -= 16
960         $FileSize -= $BytesPerCluster * 16
961         $ProgressSize = $FileSize
962     WEnd
963     If $i = 0 Or $FileSize = 0 Then Return $FileSize
964     If $i > 16 Then $i = 16
965     _WinAPI_ReadFile($hDisk, DllStructGetPtr($cBuffer), $BytesPerCluster * $i, $nBytes)
966     If $FileSize > $BytesPerCluster * $i Then
967         _WinAPI_WriteFile($hFile, DllStructGetPtr($cBuffer), $BytesPerCluster * $i, $nBytes)
968         $FileSize -= $BytesPerCluster * $i
969         $ProgressSize = $FileSize
970         Return $FileSize
971     Else
972         _WinAPI_WriteFile($hFile, DllStructGetPtr($cBuffer), $FileSize, $nBytes)
973         $ProgressSize = 0
974         Return 0
975     EndIf
976 EndFunc
977
978 Func _DoSparse($r,$hFile,$FileSize)
979     Local $nBytes
980     If Not IsDllStruct($sBuffer) Then _CreateSparseBuffer()
981     $i = $RUN_Clusters[$r]
982     While $i > 16 And $FileSize > $BytesPerCluster * 16
983         _WinAPI_WriteFile($hFile, DllStructGetPtr($sBuffer), $BytesPerCluster * 16, $nBytes)
984
985         $i -= 16
986         $FileSize -= $BytesPerCluster * 16
987         $ProgressSize = $FileSize
988     WEnd
989     If $i <> 0 Then
990         If $FileSize > $BytesPerCluster * $i Then
991             _WinAPI_WriteFile($hFile, DllStructGetPtr($sBuffer), $BytesPerCluster * $i, $nBytes)
992             $FileSize -= $BytesPerCluster * $i
993             $ProgressSize = $FileSize
994         Else
995             _WinAPI_WriteFile($hFile, DllStructGetPtr($sBuffer), $FileSize, $nBytes)
996             $ProgressSize = 0
997             Return 0
998         EndIf
999     EndIf
1000     Return $FileSize
1001 EndFunc
1002
1003 Func _CreateSparseBuffer()
1004     Global $sBuffer = DllStructCreate("byte[" & $BytesPerCluster * 16 & "]")
1005     For $i = 1 To $BytesPerCluster * 16
1006         DllStructSetData($sBuffer, $i, 0)
1007     Next
1008 EndFunc
1009
1010 Func _LZNTDecompress($tInput, $Size) ;note function returns a null string if error, or an array if no error
1011     Local $tOutput[2]
1012     Local $cBuffer = DllStructCreate("byte[" & $BytesPerCluster*16 & "]")
1013     Local $a_Call = DllCall("ntdll.dll", "int", "RtlDecompressBuffer", _
1014         "ushort", 2, _
1015         "ptr", DllStructGetPtr($cBuffer), _

```

```

1015     "dword", DllStructGetSize($cBuffer), _
1016     "ptr", DllStructGetPtr($tInput), _
1017     "dword", $Size, _
1018     "dword*", 0)
1019
1020 If @error Or $a_Call[0] Then           ;if $a_Call[0]=0 then output size is in $a_Call[6], otherwise $a_Call[6] is invalid
1021     Return SetError(1, 0, "") ; error decompressing
1022 EndIf
1023 Local $Decompressed = DllStructCreate("byte[" & $a_Call[6] & "]", DllStructGetPtr($cBuffer))
1024     $tOutput[0] = DllStructGetData($Decompressed, 1)
1025     $tOutput[1] = $a_Call[6]
1026     Return SetError(0, 0, $tOutput)
1027 EndFunc
1028
1029 Func _ExtractResidentFile($Name, $Size, $record)
1030     Local $nBytes
1031     $xBuffer = DllStructCreate("byte[" & $Size & "]")
1032     DllStructSetData($xBuffer, 1, '0x' & $DataRun)
1033     $hFile = _WinAPI_CreateFile($Name,3,6,7)
1034
1035     If $hFile Then
1036         _WinAPI_SetFilePointer($hFile, 0,$FILE_BEGIN)
1037         _WinAPI_WriteFile($hFile, DllStructGetPtr($xBuffer), $Size, $nBytes)
1038         _WinAPI_CloseHandle($hFile)
1039         Return
1040     Else
1041         ConsoleWrite("Error" & @CRLF)
1042     EndIf
1043 EndFunc
1044
1045 Func _TranslateAttributeType($input)
1046     Local $RetVal
1047     Select
1048         Case $input = $STANDARD_INFORMATION
1049             $RetVal = "$STANDARD_INFORMATION"
1050         Case $input = $ATTRIBUTE_LIST
1051             $RetVal = "$ATTRIBUTE_LIST"
1052         Case $input = $FILE_NAME
1053             $RetVal = "$FILE_NAME"
1054         Case $input = $OBJECT_ID
1055             $RetVal = "$OBJECT_ID"
1056         Case $input = $SECURITY_DESCRIPTOR
1057             $RetVal = "$SECURITY_DESCRIPTOR"
1058         Case $input = $VOLUME_NAME
1059             $RetVal = "$VOLUME_NAME"
1060         Case $input = $VOLUME_INFORMATION
1061             $RetVal = "$VOLUME_INFORMATION"
1062         Case $input = $DATA
1063             $RetVal = "$DATA"
1064         Case $input = $INDEX_ROOT
1065             $RetVal = "$INDEX_ROOT"
1066         Case $input = $INDEX_ALLOCATION
1067             $RetVal = "$INDEX_ALLOCATION"
1068         Case $input = $BITMAP
1069             $RetVal = "$BITMAP"
1070         Case $input = $REPARSE_POINT
1071             $RetVal = "$REPARSE_POINT"
1072         Case $input = $EA_INFORMATION
1073             $RetVal = "$EA_INFORMATION"
1074         Case $input = $EA
1075             $RetVal = "$EA"
1076         Case $input = $PROPERTY_SET
1077             $RetVal = "$PROPERTY_SET"
1078         Case $input = $LOGGED_UTILITY_STREAM
1079             $RetVal = "$LOGGED_UTILITY_STREAM"
1080         Case $input = $ATTRIBUTE_END_MARKER
1081             $RetVal = "$ATTRIBUTE_END_MARKER"
1082     EndSelect
1083     Return $RetVal
1084 EndFunc
1085
1086 Func NT_SUCCESS($status)
1087     If 0 <= $status And $status <= 0x7FFFFFFF Then
1088         Return True

```



```
1088 Else
1089     Return False
1090 EndIf
1091 EndFunc
1092
1093 Func _GetAttributeEntry($Entry)
1094     Local $CoreAttribute,$CoreAttributeTmp,$CoreAttributeArr[2]
1095     Local $ATTRIBUTE_HEADER_Length,$ATTRIBUTE_HEADER_NonResidentFlag,$ATTRIBUTE_HEADER_NameLength,$ATTRIBUTE_HEADER_Name
1096     Local $ATTRIBUTE_HEADER_VCNS,$ATTRIBUTE_HEADER_OffsetToDataRuns,$ATTRIBUTE_HEADER_CompressionUnitSize,$ATTRIBUTE_HEADER_OffsetToAttribute,$ATTRIBUTE_HEADER_IndexedFlag
1097     Local $ATTRIBUTE_HEADER_LengthOfAttribute,$ATTRIBUTE_HEADER_OffsetToAttribute,$ATTRIBUTE_HEADER_IndexedFlag
1098     $ATTRIBUTE_HEADER_Length = StringMid($Entry,9,8)
1099     $ATTRIBUTE_HEADER_Length = Dec(StringMid($ATTRIBUTE_HEADER_Length,7,2) & StringMid($ATTRIBUTE_HEADER_Length,5,2) & StringMid($ATTRIBUTE_HEADER_Length,3,2))
1100     $ATTRIBUTE_HEADER_NonResidentFlag = StringMid($Entry,17,2)
1101 ; ConsoleWrite("ATTRIBUTE_HEADER_NonResidentFlag = " & $ATTRIBUTE_HEADER_NonResidentFlag & @crlf)
1102     $ATTRIBUTE_HEADER_NameLength = Dec(StringMid($Entry,19,2))
1103 ; ConsoleWrite("ATTRIBUTE_HEADER_NameLength = " & $ATTRIBUTE_HEADER_NameLength & @crlf)
1104     $ATTRIBUTE_HEADER_NameRelativeOffset = StringMid($Entry,21,4)
1105 ; ConsoleWrite("ATTRIBUTE_HEADER_NameRelativeOffset = " & $ATTRIBUTE_HEADER_NameRelativeOffset & @crlf)
1106     $ATTRIBUTE_HEADER_NameRelativeOffset = Dec(_SwapEndian($ATTRIBUTE_HEADER_NameRelativeOffset))
1107 ; ConsoleWrite("ATTRIBUTE_HEADER_NameRelativeOffset = " & $ATTRIBUTE_HEADER_NameRelativeOffset & @crlf)
1108     If $ATTRIBUTE_HEADER_NameLength > 0 Then
1109         $ATTRIBUTE_HEADER_Name = _UnicodeHexToStr(StringMid($Entry,$ATTRIBUTE_HEADER_NameRelativeOffset*2 + 1,$ATTRIBUTE_HEADER_NameLength))
1110     Else
1111         $ATTRIBUTE_HEADER_Name = ""
1112     EndIf
1113     $ATTRIBUTE_HEADER_Flags = _SwapEndian(StringMid($Entry,25,4))
1114 ; ConsoleWrite("ATTRIBUTE_HEADER_Flags = " & $ATTRIBUTE_HEADER_Flags & @crlf)
1115     $Flags = ""
1116     If $ATTRIBUTE_HEADER_Flags = "0000" Then
1117         $Flags = "NORMAL"
1118     Else
1119         If BitAND($ATTRIBUTE_HEADER_Flags,"0001") Then
1120             $IsCompressed = 1
1121             $Flags &= "COMPRESSED+"
1122         EndIf
1123         If BitAND($ATTRIBUTE_HEADER_Flags,"4000") Then
1124             $IsEncrypted = 1
1125             $Flags &= "ENCRYPTED+"
1126         EndIf
1127         If BitAND($ATTRIBUTE_HEADER_Flags,"8000") Then
1128             $IsSparse = 1
1129             $Flags &= "SPARSE+"
1130         EndIf
1131         $Flags = StringTrimRight($Flags,1)
1132     EndIf
1133 ; ConsoleWrite("File is " & $Flags & @CRLF)
1134
1135     $ATTRIBUTE_HEADER_AttributeID = StringMid($Entry,29,4)
1136     $ATTRIBUTE_HEADER_AttributeID = StringMid($ATTRIBUTE_HEADER_AttributeID,3,2) & StringMid($ATTRIBUTE_HEADER_AttributeID,1,2)
1137     If $ATTRIBUTE_HEADER_NonResidentFlag = '01' Then
1138         $ATTRIBUTE_HEADER_StartVCN = StringMid($Entry,33,16)
1139 ; ConsoleWrite("ATTRIBUTE_HEADER_StartVCN = " & $ATTRIBUTE_HEADER_StartVCN & @crlf)
1140         $ATTRIBUTE_HEADER_StartVCN = Dec(_SwapEndian($ATTRIBUTE_HEADER_StartVCN),2)
1141 ; ConsoleWrite("ATTRIBUTE_HEADER_StartVCN = " & $ATTRIBUTE_HEADER_StartVCN & @crlf)
1142         $ATTRIBUTE_HEADER_LastVCN = StringMid($Entry,49,16)
1143 ; ConsoleWrite("ATTRIBUTE_HEADER_LastVCN = " & $ATTRIBUTE_HEADER_LastVCN & @crlf)
1144         $ATTRIBUTE_HEADER_LastVCN = Dec(_SwapEndian($ATTRIBUTE_HEADER_LastVCN),2)
1145 ; ConsoleWrite("ATTRIBUTE_HEADER_LastVCN = " & $ATTRIBUTE_HEADER_LastVCN & @crlf)
1146         $ATTRIBUTE_HEADER_VCNS = $ATTRIBUTE_HEADER_LastVCN - $ATTRIBUTE_HEADER_StartVCN
1147 ; ConsoleWrite("ATTRIBUTE_HEADER_VCNS = " & $ATTRIBUTE_HEADER_VCNS & @crlf)
1148         $ATTRIBUTE_HEADER_OffsetToDataRuns = StringMid($Entry,65,4)
1149         $ATTRIBUTE_HEADER_OffsetToDataRuns = Dec(StringMid($ATTRIBUTE_HEADER_OffsetToDataRuns,3,1) & StringMid($ATTRIBUTE_HEADER_OffsetToDataRuns,1,3))
1150         $ATTRIBUTE_HEADER_CompressionUnitSize = Dec(_SwapEndian(StringMid($Entry,69,4)))
1151 ; ConsoleWrite("ATTRIBUTE_HEADER_CompressionUnitSize = " & $ATTRIBUTE_HEADER_CompressionUnitSize & @crlf)
1152         $IsCompressed = 0
1153         If $ATTRIBUTE_HEADER_CompressionUnitSize = 4 Then $IsCompressed = 1
1154         $ATTRIBUTE_HEADER_Padding = StringMid($Entry,73,8)
1155         $ATTRIBUTE_HEADER_Padding = StringMid($ATTRIBUTE_HEADER_Padding,7,2) & StringMid($ATTRIBUTE_HEADER_Padding,1,7)
1156         $ATTRIBUTE_HEADER_AllocatedSize = StringMid($Entry,81,16)
1157 ; ConsoleWrite("ATTRIBUTE_HEADER_AllocatedSize = " & $ATTRIBUTE_HEADER_AllocatedSize & @crlf)
1158         $ATTRIBUTE_HEADER_AllocatedSize = Dec(_SwapEndian($ATTRIBUTE_HEADER_AllocatedSize),2)
1159 ; ConsoleWrite("ATTRIBUTE_HEADER_AllocatedSize = " & $ATTRIBUTE_HEADER_AllocatedSize & @crlf)
1160         $ATTRIBUTE_HEADER_RealSize = StringMid($Entry,97,16)
1161 ; ConsoleWrite("ATTRIBUTE_HEADER_RealSize = " & $ATTRIBUTE_HEADER_RealSize & @crlf)
1162         $ATTRIBUTE_HEADER_RealSize = Dec(_SwapEndian($ATTRIBUTE_HEADER_RealSize),2)
```

```

1162 ; ConsoleWrite("$ATTRIBUTE_HEADER_RealSize = " & $ATTRIBUTE_HEADER_RealSize & @crlf)
1163 $ATTRIBUTE_HEADER_InitializedStreamSize = StringMid($Entry,113,16)
1164 ; ConsoleWrite("$ATTRIBUTE_HEADER_InitializedStreamSize = " & $ATTRIBUTE_HEADER_InitializedStreamSize & @crlf)
1165 $ATTRIBUTE_HEADER_InitializedStreamSize = Dec(_SwapEndian($ATTRIBUTE_HEADER_InitializedStreamSize),2)
1166 ; ConsoleWrite("$ATTRIBUTE_HEADER_InitializedStreamSize = " & $ATTRIBUTE_HEADER_InitializedStreamSize & @crlf)
1167 $RunListOffset = StringMid($Entry,65,4)
1168 ; ConsoleWrite("$RunListOffset = " & $RunListOffset & @crlf)
1169 $RunListOffset = Dec(_SwapEndian($RunListOffset))
1170 ; ConsoleWrite("$RunListOffset = " & $RunListOffset & @crlf)
1171 If $IsCompressed AND $RunListOffset = 72 Then
1172     $ATTRIBUTE_HEADER_CompressedSize = StringMid($Entry,129,16)
1173     $ATTRIBUTE_HEADER_CompressedSize = Dec(_SwapEndian($ATTRIBUTE_HEADER_CompressedSize),2)
1174 EndIf
1175 $DataRun = StringMid($Entry,$RunListOffset*2+1,(StringLen($Entry)-$RunListOffset)*2)
1176 ; ConsoleWrite("$DataRun = " & $DataRun & @crlf)
1177 ElseIf $ATTRIBUTE_HEADER_NonResidentFlag = '00' Then
1178     $ATTRIBUTE_HEADER_LengthOfAttribute = StringMid($Entry,33,8)
1179 ; ConsoleWrite("$ATTRIBUTE_HEADER_LengthOfAttribute = " & $ATTRIBUTE_HEADER_LengthOfAttribute & @crlf)
1180 $ATTRIBUTE_HEADER_LengthOfAttribute = Dec(_SwapEndian($ATTRIBUTE_HEADER_LengthOfAttribute),2)
1181 ; ConsoleWrite("$ATTRIBUTE_HEADER_LengthOfAttribute = " & $ATTRIBUTE_HEADER_LengthOfAttribute & @crlf)
1182 ; $ATTRIBUTE_HEADER_OffsetToAttribute = StringMid($Entry,41,4)
1183 ; $ATTRIBUTE_HEADER_OffsetToAttribute = Dec(StringMid($ATTRIBUTE_HEADER_OffsetToAttribute,3,2) & StringMid($A
1184
1185     $ATTRIBUTE_HEADER_OffsetToAttribute = Dec(_SwapEndian(StringMid($Entry,41,4)))
1186 ; ConsoleWrite("$ATTRIBUTE_HEADER_OffsetToAttribute = " & $ATTRIBUTE_HEADER_OffsetToAttribute & @crlf)
1187 $ATTRIBUTE_HEADER_IndexedFlag = Dec(StringMid($Entry,45,2))
1188 $ATTRIBUTE_HEADER_Padding = StringMid($Entry,47,2)
1189 $DataRun = StringMid($Entry,$ATTRIBUTE_HEADER_OffsetToAttribute*2+1,$ATTRIBUTE_HEADER_LengthOfAttribute*2)
1190 ; ConsoleWrite("$DataRun = " & $DataRun & @crlf)
1191 EndIf
1192 ; Possible continuation
1193 For $i = 1 To UBound($DataQ) - 1
1194     For $i = 1 To 1
1195         _DecodeDataQEntry($DataQ[$i])
1196         If $ATTRIBUTE_HEADER_NonResidentFlag = '00' Then
1197             ;_ExtractResidentFile($DATA_Name, $DATA_LengthOfAttribute)
1198             $CoreAttribute = $DataRun
1199             Else
1200                 Global $RUN_VCN[1], $RUN_Clusters[1]
1201
1202                 $TotalClusters = $ATTRIBUTE_HEADER_LastVCN - $ATTRIBUTE_HEADER_StartVCN + 1
1203                 $Size = $ATTRIBUTE_HEADER_RealSize
1204 ;_ExtractDataRuns()
1205 $r=UBound($RUN_Clusters)
1206 $i=1
1207 $RUN_VCN[0] = 0
1208 $BaseVCN = $RUN_VCN[0]
1209 If $DataRun = "" Then $DataRun = "00"
1210 Do
1211     $RunListID = StringMid($DataRun,$i,2)
1212     If $RunListID = "00" Then ExitLoop
1213 ; ConsoleWrite("$RunListID = " & $RunListID & @crlf)
1214 $i += 2
1215 $RunListClustersLength = Dec(StringMid($RunListID,2,1))
1216 ; ConsoleWrite("$RunListClustersLength = " & $RunListClustersLength & @crlf)
1217 $RunListVCNLength = Dec(StringMid($RunListID,1,1))
1218 ; ConsoleWrite("$RunListVCNLength = " & $RunListVCNLength & @crlf)
1219 $RunListClusters = Dec(_SwapEndian(StringMid($DataRun,$i,$RunListClustersLength*2)),2)
1220 ; ConsoleWrite("$RunListClusters = " & $RunListClusters & @crlf)
1221 $i += $RunListClustersLength*2
1222 $RunListVCN = _SwapEndian(StringMid($DataRun, $i, $RunListVCNLength*2))
1223 ;next line handles positive or negative move
1224 $BaseVCN += Dec($RunListVCN,2)-((($r>1) And (Dec(StringMid($RunListVCN,1,1))>7))*Dec(StringM
1225 If $RunListVCN <> "" Then
1226     $RunListVCN = $BaseVCN
1227 Else
1228     $RunListVCN = 0 ;$RUN_VCN[$r-1] ;0
1229 EndIf
1230 ; ConsoleWrite("$RunListVCN = " & $RunListVCN & @crlf)
1231 If (($RunListVCN=0) And ($RunListClusters>16) And (Mod($RunListClusters,16)>0)) Then
1232 ;If (($RunListVCN=$RUN_VCN[$r-1]) And ($RunListClusters>16) And (Mod($RunListClusters,16)>0)
1233 ;may be sparse section at end of Compression Signature
1234 _ArrayAdd($RUN_Clusters,Mod($RunListClusters,16))
1235
1236 _ArrayAdd($RUN_VCN,$RunListVCN)

```

```

1235 $RunListClusters -= Mod($RunListClusters,16)
1236 $r += 1
1237 ElseIf (($RunListClusters>16) And (Mod($RunListClusters,16)>0)) Then
1238 ;may be compressed data section at start of Compression Signature
1239 _ArrayAdd($RUN_Clusters,$RunListClusters-Mod($RunListClusters,16))
1240 _ArrayAdd($RUN_VCN,$RunListVCN)
1241 $RunListVCN += $RUN_Clusters[$r]
1242 $RunListClusters = Mod($RunListClusters,16)
1243 $r += 1
1244 EndIf
1245 ;just normal or sparse data
1246 _ArrayAdd($RUN_Clusters,$RunListClusters)
1247 _ArrayAdd($RUN_VCN,$RunListVCN)
1248 $r += 1
1249 $i += $RunListVCNLength*2
1250 Until $i > StringLen($DataRun)
1251 ;-----_ExtractDataRuns()
1252 ; _ArrayDisplay($RUN_Clusters,"$RUN_Clusters")
1253 ; _ArrayDisplay($RUN_VCN,"$RUN_VCN")
1254 If $TotalClusters * $BytesPerCluster >= $Size Then
1255 ; ConsoleWrite(_ArrayToString($RUN_VCN) & @CRLF)
1256 ; ConsoleWrite(_ArrayToString($RUN_Clusters) & @CRLF)
1257 ;ExtractFile
1258 Local $nBytes
1259 $hFile = _WinAPI_CreateFile("\\.\\" & $TargetDrive, 2, 6, 6)
1260 If $hFile = 0 Then
1261 ConsoleWrite("Error in function _WinAPI_CreateFile when trying to open target drive
1262 _WinAPI_CloseHandle($hFile)
1263 Return
1264 EndIf
1265 $tBuffer = DllStructCreate("byte[" & $BytesPerCluster * 16 & "]")
1266 Select
1267 Case UBound($RUN_VCN) = 1 ;no data, do nothing
1268 Case (UBound($RUN_VCN) = 2) Or (Not $IsCompressed) ;may be normal or sparse
1269 If $RUN_VCN[1] = $RUN_VCN[0] And $DATA_Name <> "$Boot" Then ;sp
1270 ; _DoSparse($hTest)
1271 ConsoleWrite("Error: Sparse attributes not supported!!!" & @CRLF)
1272 Else ;normal
1273 ; _DoNormalAttribute($hFile, $tBuffer)
1274 ; Local $nBytes
1275 $FileSize = $ATTRIBUTE_HEADER_RealSize
1276 For $s = 1 To UBound($RUN_VCN)-1
1277 _WinAPI_SetFilePointerEx($hFile, $RUN_VCN[$s]*$BytesPerCluster,
1278 $g = $RUN_Clusters[$s]
1279 While $g > 16 And $FileSize > $BytesPerCluster * 16
1280 _WinAPI_ReadFile($hFile, DllStructGetPtr($tBuffer),
1281 ; _WinAPI_WriteFile($hTest, DllStructGetPtr($tBuffer),
1282 $g -= 16
1283 $FileSize -= $BytesPerCluster * 16
1284 $CoreAttributeTmp = StringMid(DllStructGetData($tBuffer, $g), 1, $BytesPerCluster)
1285 $CoreAttribute &= $CoreAttributeTmp
1286 WEnd
1287 If $g <> 0 Then
1288 _WinAPI_ReadFile($hFile, DllStructGetPtr($tBuffer),
1289 $CoreAttributeTmp = StringMid(DllStructGetData($tBuffer, $g), 1, $BytesPerCluster)
1290 $CoreAttribute &= $CoreAttributeTmp
1291 If $FileSize > $BytesPerCluster * $g Then
1292 _WinAPI_WriteFile($hTest, DllStructGetPtr($tBuffer), $g, $CoreAttributeTmp)
1293 $FileSize -= $BytesPerCluster * $g
1294 $CoreAttributeTmp = StringMid(DllStructGetData($tBuffer, $g), 1, $BytesPerCluster)
1295 $CoreAttribute &= $CoreAttributeTmp
1296 Else
1297 _WinAPI_WriteFile($hTest, DllStructGetPtr($tBuffer), $g, $CoreAttributeTmp)
1298 Return
1299 $CoreAttributeTmp = StringMid(DllStructGetData($tBuffer, $g), 1, $BytesPerCluster)
1300 $CoreAttribute &= $CoreAttributeTmp
1301 EndIf
1302 EndIf
1303 Next
1304 ;-----_DoNormalAttribute()
1305 EndIf
1306 Case Else ;may be compressed
1307 ; _DoCompressed($hFile, $hTest, $tBuffer)
1308 ConsoleWrite("Error: Compressed attributes not supported!!!" & @CRLF)
1309 EndIf

```

```
1309         EndSelect
1310     ;-----ExtractFile
1311         EndIf
1312     ;-----
1313         EndIf
1314     Next
1315     $CoreAttributeArr[0] = $CoreAttribute
1316     $CoreAttributeArr[1] = $ATTRIBUTE_HEADER_Name
1317     Return $CoreAttributeArr
1318 EndFunc
1319
1320 Func _Get_IndexRoot($Entry,$Current_Attrib_Number,$CurrentAttributeName)
1321     Local $LocalAttributeOffset = 1,$AttributeType,$CollationRule,$SizeOfIndexAllocationEntry,$ClustersPerIndexRoot,$IR
1322     $AttributeType = StringMid($Entry,$LocalAttributeOffset,8)
1323     ; $AttributeType = _SwapEndian($AttributeType)
1324     $CollationRule = StringMid($Entry,$LocalAttributeOffset+8,8)
1325     $CollationRule = _SwapEndian($CollationRule)
1326     $SizeOfIndexAllocationEntry = StringMid($Entry,$LocalAttributeOffset+16,8)
1327     $SizeOfIndexAllocationEntry = Dec(_SwapEndian($SizeOfIndexAllocationEntry),2)
1328     $ClustersPerIndexRoot = Dec(StringMid($Entry,$LocalAttributeOffset+24,2))
1329     ; $IRPadding = StringMid($Entry,$LocalAttributeOffset+26,6)
1330     $OffsetToFirstEntry = StringMid($Entry,$LocalAttributeOffset+32,8)
1331     $OffsetToFirstEntry = Dec(_SwapEndian($OffsetToFirstEntry),2)
1332     $TotalSizeOfEntries = StringMid($Entry,$LocalAttributeOffset+40,8)
1333     $TotalSizeOfEntries = Dec(_SwapEndian($TotalSizeOfEntries),2)
1334
1335     $AllocatedSizeOfEntries = StringMid($Entry,$LocalAttributeOffset+48,8)
1336     $AllocatedSizeOfEntries = Dec(_SwapEndian($AllocatedSizeOfEntries),2)
1337     $Flags = StringMid($Entry,$LocalAttributeOffset+56,2)
1338     If $Flags = "01" Then
1339         $Flags = "01 (Index Allocation needed)"
1340         $ResidentIdx = 0
1341     Else
1342         $Flags = "00 (Fits in Index Root)"
1343         $ResidentIdx = 1
1344     EndIf
1345     ; $IRPadding2 = StringMid($Entry,$LocalAttributeOffset+58,6)
1346     $IRArr[0][$Current_Attrib_Number] = "IndexRoot Number " & $Current_Attrib_Number
1347     $IRArr[1][$Current_Attrib_Number] = $CurrentAttributeName
1348     $IRArr[2][$Current_Attrib_Number] = $AttributeType
1349     $IRArr[3][$Current_Attrib_Number] = $CollationRule
1350     $IRArr[4][$Current_Attrib_Number] = $SizeOfIndexAllocationEntry
1351     $IRArr[5][$Current_Attrib_Number] = $ClustersPerIndexRoot
1352     ; $IRArr[6][$Current_Attrib_Number] = $IRPadding
1353     $IRArr[7][$Current_Attrib_Number] = $OffsetToFirstEntry
1354     $IRArr[8][$Current_Attrib_Number] = $TotalSizeOfEntries
1355     $IRArr[9][$Current_Attrib_Number] = $AllocatedSizeOfEntries
1356     $IRArr[10][$Current_Attrib_Number] = $Flags
1357     ; $IRArr[11][$Current_Attrib_Number] = $IRPadding2
1358     $TheResidentIndexEntry = StringMid($Entry,$LocalAttributeOffset+64)
1359     If $ResidentIdx And $AttributeType=FILE_NAME Then
1360         ; $TheResidentIndexEntry = StringMid($Entry,$LocalAttributeOffset+64)
1361         _DecodeIdxEntries($TheResidentIndexEntry,$Current_Attrib_Number,$CurrentAttributeName)
1362     ElseIf $ResidentIdx=0 And $AttributeType=FILE_NAME Then
1363         _DecodeIdxEntries($TheResidentIndexEntry,$Current_Attrib_Number,$CurrentAttributeName)
1364     EndIf
1365 EndFunc
1366
1367 Func _StripIdxRecord($Entry)
1368     ; ConsoleWrite("Starting function _StripIdxRecord()" & @CrLf)
1369     Local $LocalAttributeOffset = 1,$IdxHdrUpdateSeqArrOffset,$IdxHdrUpdateSeqArrSize,$IdxHdrUpdSeqArr,$IdxHdrUpdSe
1370     Local $IdxRecordEnd1,$IdxRecordEnd2,$IdxRecordEnd3,$IdxRecordEnd4,$IdxRecordEnd5,$IdxRecordEnd6,$IdxRecordEnd
1371     ; ConsoleWrite("Unfixed IDX record:" & @CrLf)
1372     ; ConsoleWrite(_HexEncode("0x"&$Entry) & @CrLf)
1373     ; ConsoleWrite(_HexEncode("0x" & StringMid($Entry,1,4096)) & @CrLf)
1374     $IdxHdrUpdateSeqArrOffset = Dec(_SwapEndian(StringMid($Entry,$LocalAttributeOffset+8,4)))
1375     ; ConsoleWrite("$IdxHdrUpdateSeqArrOffset = " & $IdxHdrUpdateSeqArrOffset & @CrLf)
1376     $IdxHdrUpdateSeqArrSize = Dec(_SwapEndian(StringMid($Entry,$LocalAttributeOffset+12,4)))
1377     ; ConsoleWrite("$IdxHdrUpdateSeqArrSize = " & $IdxHdrUpdateSeqArrSize & @CrLf)
1378     $IdxHdrUpdSeqArr = StringMid($Entry,1+($IdxHdrUpdateSeqArrOffset*2),$IdxHdrUpdateSeqArrSize*2*2)
1379     ; ConsoleWrite("$IdxHdrUpdSeqArr = " & $IdxHdrUpdSeqArr & @CrLf)
1380     $IdxHdrUpdSeqArrPart0 = StringMid($IdxHdrUpdSeqArr,1,4)
1381     $IdxHdrUpdSeqArrPart1 = StringMid($IdxHdrUpdSeqArr,5,4)
1382     $IdxHdrUpdSeqArrPart2 = StringMid($IdxHdrUpdSeqArr,9,4)
1383     $IdxHdrUpdSeqArrPart3 = StringMid($IdxHdrUpdSeqArr,13,4)
1384     $IdxHdrUpdSeqArrPart4 = StringMid($IdxHdrUpdSeqArr,17,4)
```

```

1382 $IndxHdrUpdSeqArrPart4 = StringMid($IndxHdrUpdSeqArr,17,4)
1384 $IndxHdrUpdSeqArrPart5 = StringMid($IndxHdrUpdSeqArr,21,4)
1385 $IndxHdrUpdSeqArrPart6 = StringMid($IndxHdrUpdSeqArr,25,4)
1386 $IndxHdrUpdSeqArrPart7 = StringMid($IndxHdrUpdSeqArr,29,4)
1387 $IndxHdrUpdSeqArrPart8 = StringMid($IndxHdrUpdSeqArr,33,4)
1388 $IndxRecordEnd1 = StringMid($Entry,1021,4)
1389 $IndxRecordEnd2 = StringMid($Entry,2045,4)
1390 $IndxRecordEnd3 = StringMid($Entry,3069,4)
1391 $IndxRecordEnd4 = StringMid($Entry,4093,4)
1392 $IndxRecordEnd5 = StringMid($Entry,5117,4)
1393 $IndxRecordEnd6 = StringMid($Entry,6141,4)
1394 $IndxRecordEnd7 = StringMid($Entry,7165,4)
1395 $IndxRecordEnd8 = StringMid($Entry,8189,4)
1396 If $IndxHdrUpdSeqArrPart0 <> $IndxRecordEnd1 OR $IndxHdrUpdSeqArrPart0 <> $IndxRecordEnd2 OR $IndxHdrUpdSeqArrPart0
1397     ConsoleWrite("Error the INDX record is corrupt" & @CRLF)
1398     Return ; Not really correct because I think in theory chunks of 1024 bytes can be invalid and not just ever.
1399 Else
1400     $Entry = StringMid($Entry,1,1020) & $IndxHdrUpdSeqArrPart1 & StringMid($Entry,1025,1020) & $IndxHdrUpdSeqAr
1401 EndIf
1402 $IndxRecordSize = Dec(_SwapEndian(StringMid($Entry,$LocalAttributeOffset+56,8)),2)
1403 ; ConsoleWrite("$IndxRecordSize = " & $IndxRecordSize & @CRLF)
1404 $IndxHeaderSize = Dec(_SwapEndian(StringMid($Entry,$LocalAttributeOffset+48,8)),2)
1405 ; ConsoleWrite("$IndxHeaderSize = " & $IndxHeaderSize & @CRLF)
1406 $IsNotLeafNode = StringMid($Entry,$LocalAttributeOffset+72,2) ;1 if not Leaf node
1407 $Entry = StringMid($Entry,$LocalAttributeOffset+48+($IndxHeaderSize*2),($IndxRecordSize-$IndxHeaderSize-16)*2)
1408 If $IsNotLeafNode = "01" Then ; This flag leads to the entry being 8 bytes of 00's Longer than the others. Can be
1409     $Entry = StringTrimRight($Entry,16)
1410 ; ConsoleWrite("Is not Leaf node..." & @CRLF)
1411 EndIf
1412 Return $Entry
1413 EndFunc

1415 Func _Get_IndexAllocation($Entry,$Current_Attrib_Number,$CurrentAttributeName)
1416 ; ConsoleWrite("Starting function _Get_IndexAllocation()" & @CRLF)
1417 Local $NextPosition = 1,$IndxHdrMagic,$IndxEntries,$TotalIndxEntries
1418 ; ConsoleWrite("StringLen of chunk = " & StringLen($Entry) & @CRLF)
1419 ; ConsoleWrite("Expected records = " & StringLen($Entry)/8192 & @CRLF)
1420 $NextPosition = 1
1421 Do
1422     $IndxHdrMagic = StringMid($Entry,$NextPosition,8)
1423 ; ConsoleWrite("$IndxHdrMagic = " & $IndxHdrMagic & @CRLF)
1424 $IndxHdrMagic = _HexToString($IndxHdrMagic)
1425 ; ConsoleWrite("$IndxHdrMagic = " & $IndxHdrMagic & @CRLF)
1426 If $IndxHdrMagic <> "INDX" Then
1427 ; ConsoleWrite("$IndxHdrMagic: " & $IndxHdrMagic & @CRLF)
1428 ; ConsoleWrite("Error: Record is not of type INDX, and this was not expected.." & @CRLF)
1429     $NextPosition += 8192
1430     ContinueLoop
1431 EndIf
1432 $IndxEntries = _StripIndxRecord(StringMid($Entry,$NextPosition,8192))
1433 $TotalIndxEntries &= $IndxEntries

1434     $NextPosition += 8192
1435 Until $NextPosition >= StringLen($Entry)+32
1436 ; ConsoleWrite("INDX record:" & @CRLF)
1437 ; ConsoleWrite(_HexEncode("0x"&StringMid($Entry,1)) & @CRLF)
1438 ; ConsoleWrite("Total chunk of stripped INDX entries:" & @CRLF)
1439 ; ConsoleWrite(_HexEncode("0x"&StringMid($TotalIndxEntries,1)) & @CRLF)
1440 _DecodeIndxEntries($TotalIndxEntries,$Current_Attrib_Number,$CurrentAttributeName)
1441 EndFunc

1443 Func _DecodeIndxEntries($Entry,$Current_Attrib_Number,$CurrentAttributeName)
1444 ; ConsoleWrite("Starting function _DecodeIndxEntries()" & @CRLF)
1445 Local $LocalAttributeOffset = 1,$NewLocalAttributeOffset,$IndxHdrMagic,$IndxHdrUpdateSeqArrOffset,$IndxHdrUpdateSeq
1446 Local $IndxHdrFlag,$IndxHdrPadding,$IndxHdrUpdateSequence,$IndxHdrUpdSeqArr,$IndxHdrUpdSeqArrPart0,$IndxHdrUpdSeqAr
1447 Local $FileReference,$IndexEntryLength,$StreamLength,$Flags,$Stream,$SubNodeVCN,$tmp0=0,$tmp1=0,$tmp2=0,$tmp3=0,$En
1448 $NewLocalAttributeOffset = 1
1449 $MFTReference = StringMid($Entry,$NewLocalAttributeOffset,12)
1450 $MFTReference = StringMid($MFTReference,7,2)&StringMid($MFTReference,5,2)&StringMid($MFTReference,3,2)&StringMid($M
1451 $MFTReference = Dec($MFTReference)
1452 $MFTReferenceSeqNo = StringMid($Entry,$NewLocalAttributeOffset+12,4)
1453 $MFTReferenceSeqNo = Dec(StringMid($MFTReferenceSeqNo,3,2)&StringMid($MFTReferenceSeqNo,1,2))
1454 $IndexEntryLength = StringMid($Entry,$NewLocalAttributeOffset+16,4)
1455 $IndexEntryLength = Dec(StringMid($IndexEntryLength,3,2)&StringMid($IndexEntryLength,3,2))
1456 $OffsetToFileNmae = StringMid($Entry,$NewLocalAttributeOffset+20,4)

```

```

1457 $OffsetToFileSize = StringMid($Entry,$NewLocalAttributeOffset+20,4)
1458 $OffsetToFileSize = Dec(StringMid($OffsetToFileSize,3,2)&StringMid($OffsetToFileSize,3,2))
1459 ; $IndexFlags = StringMid($Entry,$NewLocalAttributeOffset+24,4)
1460 $MFTReferenceOfParent = StringMid($Entry,$NewLocalAttributeOffset+32,12)
1461 $MFTReferenceOfParent = StringMid($MFTReferenceOfParent,7,2)&StringMid($MFTReferenceOfParent,5,2)&StringMid($MFTReferenceOfParent,5,2)
1462 $MFTReferenceOfParent = Dec($MFTReferenceOfParent)
1463 $MFTReferenceOfParentSeqNo = StringMid($Entry,$NewLocalAttributeOffset+44,4)
1464 $MFTReferenceOfParentSeqNo = Dec(StringMid($MFTReferenceOfParentSeqNo,3,2) & StringMid($MFTReferenceOfParentSeqNo,3,2))
1465 $Indx_CTime = StringMid($Entry,$NewLocalAttributeOffset+48,16)
1466 $Indx_CTime = StringMid($Indx_CTime,15,2) & StringMid($Indx_CTime,13,2) & StringMid($Indx_CTime,11,2) & StringMid($Indx_CTime,9,2)
1467 $Indx_CTime_tmp = _WinTime_UTCTimeToLocalFileTime("0x" & $Indx_CTime)
1468 $Indx_CTime = _WinTime_UTCTimeFormat(Dec($Indx_CTime)-$tDelta,$DateTimeFormat,2)
1469 If @error Then
1470 ; $Indx_CTime = "-"
1471 $Indx_CTime = "1601-01-01 00:00:00:000:0000"
1472 Else
1473 $Indx_CTime = $Indx_CTime & ":" & _FillZero(StringRight($Indx_CTime_tmp,4))
1474 EndIf
1475 $Indx_ATime = StringMid($Entry,$NewLocalAttributeOffset+64,16)
1476 $Indx_ATime = StringMid($Indx_ATime,15,2) & StringMid($Indx_ATime,13,2) & StringMid($Indx_ATime,11,2) & StringMid($Indx_ATime,9,2)
1477 $Indx_ATime_tmp = _WinTime_UTCTimeToLocalFileTime("0x" & $Indx_ATime)
1478 $Indx_ATime = _WinTime_UTCTimeFormat(Dec($Indx_ATime)-$tDelta,$DateTimeFormat,2)
1479 If @error Then
1480 ; $Indx_ATime = "-"
1481 $Indx_ATime = "1601-01-01 00:00:00:000:0000"
1482 Else
1483 $Indx_ATime = $Indx_ATime & ":" & _FillZero(StringRight($Indx_ATime_tmp,4))
1484 EndIf
1485 $Indx_MTime = StringMid($Entry,$NewLocalAttributeOffset+80,16)
1486 $Indx_MTime = StringMid($Indx_MTime,15,2) & StringMid($Indx_MTime,13,2) & StringMid($Indx_MTime,11,2) & StringMid($Indx_MTime,9,2)
1487 $Indx_MTime_tmp = _WinTime_UTCTimeToLocalFileTime("0x" & $Indx_MTime)
1488 $Indx_MTime = _WinTime_UTCTimeFormat(Dec($Indx_MTime)-$tDelta,$DateTimeFormat,2)
1489 If @error Then
1490 ; $Indx_MTime = "-"
1491 $Indx_MTime = "1601-01-01 00:00:00:000:0000"
1492 Else
1493 $Indx_MTime = $Indx_MTime & ":" & _FillZero(StringRight($Indx_MTime_tmp,4))
1494 EndIf
1495 $Indx_RTime = StringMid($Entry,$NewLocalAttributeOffset+96,16)
1496 $Indx_RTime = StringMid($Indx_RTime,15,2) & StringMid($Indx_RTime,13,2) & StringMid($Indx_RTime,11,2) & StringMid($Indx_RTime,9,2)
1497 $Indx_RTime_tmp = _WinTime_UTCTimeToLocalFileTime("0x" & $Indx_RTime)
1498 $Indx_RTime = _WinTime_UTCTimeFormat(Dec($Indx_RTime)-$tDelta,$DateTimeFormat,2)
1499 If @error Then
1500 ; $Indx_RTime = "-"
1501 $Indx_RTime = "1601-01-01 00:00:00:000:0000"
1502 Else
1503 $Indx_RTime = $Indx_RTime & ":" & _FillZero(StringRight($Indx_RTime_tmp,4))
1504 EndIf
1505 $Indx_AllocSize = StringMid($Entry,$NewLocalAttributeOffset+112,16)
1506 $Indx_AllocSize = Dec(StringMid($Indx_AllocSize,15,2) & StringMid($Indx_AllocSize,13,2) & StringMid($Indx_AllocSize,11,2) & StringMid($Indx_AllocSize,9,2))
1507 $Indx_RealSize = StringMid($Entry,$NewLocalAttributeOffset+128,16)
1508 $Indx_RealSize = Dec(StringMid($Indx_RealSize,15,2) & StringMid($Indx_RealSize,13,2) & StringMid($Indx_RealSize,11,2) & StringMid($Indx_RealSize,9,2))
1509 $Indx_File_Flags = StringMid($Entry,$NewLocalAttributeOffset+144,16)
1510 ; $Indx_File_Flags = StringMid($Indx_File_Flags,15,2) & StringMid($Indx_File_Flags,13,2) & StringMid($Indx_File_Flags,11,2) & StringMid($Indx_File_Flags,9,2)
1511 $Indx_File_Flags = StringMid(_SwapEndian($Indx_File_Flags),9,8)
1512 $Indx_File_Flags = _File_Attributes("0x" & $Indx_File_Flags)
1513 $Indx_NameLength = StringMid($Entry,$NewLocalAttributeOffset+160,2)
1514 $Indx_NameLength = Dec($Indx_NameLength)
1515 $Indx_NameSpace = StringMid($Entry,$NewLocalAttributeOffset+162,2)
1516 Select
1517 Case $Indx_NameSpace = "00" ;POSIX
1518 $Indx_NameSpace = "POSIX"
1519 Case $Indx_NameSpace = "01" ;WIN32
1520 $Indx_NameSpace = "WIN32"
1521 Case $Indx_NameSpace = "02" ;DOS
1522 $Indx_NameSpace = "DOS"
1523 Case $Indx_NameSpace = "03" ;DOS+WIN32
1524 $Indx_NameSpace = "DOS+WIN32"
1525 EndSelect
1526 $Indx_FileName = StringMid($Entry,$NewLocalAttributeOffset+164,$Indx_NameLength*2*2)
1527 $Indx_FileName = _UnicodeHexToStr($Indx_FileName)
1528 $tmp1 = 164+($Indx_NameLength*2*2)
1529 Do ; Calculate the length of the padding - 8 byte aligned
1530 $tmp2 = $tmp1/16

```



```

1531         If Not IsInt($tmp2) Then
1532             $tmp0 = 2
1533             $tmp1 += $tmp0
1534
1535             $tmp3 += $tmp0
1536         EndIf
1537     Until IsInt($tmp2)
1538     $PaddingLength = $tmp3
1539 ; $Padding2 = StringMid($Entry,$NewLocalAttributeOffset+164+($Indx_NameLength*2*2),$PaddingLength)
1540 If $IndexFlags <> "0000" Then
1541     $SubNodeVCN = StringMid($Entry,$NewLocalAttributeOffset+164+($Indx_NameLength*2*2)+$PaddingLength,16)
1542     $SubNodeVCNLength = 16
1543 Else
1544     $SubNodeVCN = "-"
1545     $SubNodeVCNLength = 0
1546 EndIf
1547 ReDim $IndxEntryNumberArr[1+$EntryCounter]
1548 ReDim $IndxMFTReferenceArr[1+$EntryCounter]
1549 ReDim $IndxMFTRefSeqNoArr[1+$EntryCounter]
1550 ReDim $IndxIndexFlagsArr[1+$EntryCounter]
1551 ReDim $IndxMFTReferenceOfParentArr[1+$EntryCounter]
1552 ReDim $IndxMFTParentRefSeqNoArr[1+$EntryCounter]
1553 ReDim $IndxCtimeArr[1+$EntryCounter]
1554 ReDim $IndxAtimeArr[1+$EntryCounter]
1555 ReDim $IndxMTimeArr[1+$EntryCounter]
1556 ReDim $IndxRTimeArr[1+$EntryCounter]
1557 ReDim $IndxAllocSizeArr[1+$EntryCounter]
1558 ReDim $IndxRealSizeArr[1+$EntryCounter]
1559 ReDim $IndxFileFlagsArr[1+$EntryCounter]
1560 ReDim $IndxFileNameArr[1+$EntryCounter]
1561 ReDim $IndxNameSpaceArr[1+$EntryCounter]
1562 ReDim $IndxSubNodeVCNArr[1+$EntryCounter]
1563 $IndxEntryNumberArr[$EntryCounter] = $EntryCounter
1564 $IndxMFTReferenceArr[$EntryCounter] = $MFTReference
1565 $IndxMFTRefSeqNoArr[$EntryCounter] = $MFTReferenceSeqNo
1566 $IndxIndexFlagsArr[$EntryCounter] = $IndexFlags
1567 $IndxMFTReferenceOfParentArr[$EntryCounter] = $MFTReferenceOfParent
1568 $IndxMFTParentRefSeqNoArr[$EntryCounter] = $MFTReferenceOfParentSeqNo
1569 $IndxCtimeArr[$EntryCounter] = $Indx_CTime
1570 $IndxAtimeArr[$EntryCounter] = $Indx_ATime
1571 $IndxMTimeArr[$EntryCounter] = $Indx_MTime
1572 $IndxRTimeArr[$EntryCounter] = $Indx_RTime
1573 $IndxAllocSizeArr[$EntryCounter] = $Indx_AllocSize
1574 $IndxRealSizeArr[$EntryCounter] = $Indx_RealSize
1575 $IndxFileFlagsArr[$EntryCounter] = $Indx_File_Flags
1576 $IndxFileNameArr[$EntryCounter] = $Indx_FileName
1577 $IndxNameSpaceArr[$EntryCounter] = $Indx_NameSpace
1578 $IndxSubNodeVCNArr[$EntryCounter] = $SubNodeVCN
1579 ; Work through the rest of the index entries
1580 $NextEntryOffset = $NewLocalAttributeOffset+164+($Indx_NameLength*2*2)+$PaddingLength+$SubNodeVCNLength
1581 If $NextEntryOffset+64 >= StringLen($Entry) Then Return
1582 Do
1583     $EntryCounter += 1
1584     ConsoleWrite("$EntryCounter = " & $EntryCounter & @crlf)
1585
1586     $MFTReference = StringMid($Entry,$NextEntryOffset,12)
1587     ConsoleWrite("$MFTReference = " & $MFTReference & @crlf)
1588     $MFTReference = StringMid($MFTReference,7,2)&StringMid($MFTReference,5,2)&StringMid($MFTReference,3,2)&StringMid($MFTReference,15,2)&StringMid($MFTReference,13,2)&StringMid($MFTReference,11,2)&StringMid($MFTReference,9,2)
1589     ConsoleWrite("$MFTReference = " & $MFTReference & @crlf)
1590     $MFTReference = Dec($MFTReference)
1591     $MFTReferenceSeqNo = StringMid($Entry,$NextEntryOffset+12,4)
1592     $MFTReferenceSeqNo = Dec(StringMid($MFTReferenceSeqNo,3,2)&StringMid($MFTReferenceSeqNo,1,2))
1593     $IndexEntryLength = StringMid($Entry,$NextEntryOffset+16,4)
1594     ConsoleWrite("$IndexEntryLength = " & $IndexEntryLength & @crlf)
1595     $IndexEntryLength = Dec(StringMid($IndexEntryLength,3,2)&StringMid($IndexEntryLength,3,2))
1596     ConsoleWrite("$IndexEntryLength = " & $IndexEntryLength & @crlf)
1597     $OffsetToFileName = StringMid($Entry,$NextEntryOffset+20,4)
1598     ConsoleWrite("$OffsetToFileName = " & $OffsetToFileName & @crlf)
1599     $OffsetToFileName = Dec(StringMid($OffsetToFileName,3,2)&StringMid($OffsetToFileName,3,2))
1600     ConsoleWrite("$OffsetToFileName = " & $OffsetToFileName & @crlf)
1601     $IndexFlags = StringMid($Entry,$NextEntryOffset+24,4)
1602     ConsoleWrite("$IndexFlags = " & $IndexFlags & @crlf)
1603     $Padding = StringMid($Entry,$NextEntryOffset+28,4)
1604     ConsoleWrite("$Padding = " & $Padding & @crlf)

```



```

1604 $MFTReferenceOfParent = StringMid($Entry,$NextEntryOffset+32,12)
1605 ; ConsoleWrite("$MFTReferenceOfParent = " & $MFTReferenceOfParent & @crlf)
1606 $MFTReferenceOfParent = StringMid($MFTReferenceOfParent,7,2)&StringMid($MFTReferenceOfParent,5,2)&StringMid
1607 ; $MFTReferenceOfParent = StringMid($MFTReferenceOfParent,15,2)&StringMid($MFTReferenceOfParent,13,2)&StringM
1608 ; ConsoleWrite("$MFTReferenceOfParent = " & $MFTReferenceOfParent & @crlf)
1609 $MFTReferenceOfParent = Dec($MFTReferenceOfParent)
1610 $MFTReferenceOfParentSeqNo = StringMid($Entry,$NextEntryOffset+44,4)
1611 $MFTReferenceOfParentSeqNo = Dec(StringMid($MFTReferenceOfParentSeqNo,3,2) & StringMid($MFTReferenceOfParentSeqNo,1,2))
1612
1613 $Indx_CTime = StringMid($Entry,$NextEntryOffset+48,16)
1614 $Indx_CTime = StringMid($Indx_CTime,15,2) & StringMid($Indx_CTime,13,2) & StringMid($Indx_CTime,11,2) & StringMid($Indx_CTime,9,2)
1615 $Indx_CTime_tmp = _WinTime_UTCTimeToLocalFileTime("0x" & $Indx_CTime)
1616 $Indx_CTime = _WinTime_UTCTimeFormat(Dec($Indx_CTime)-$tDelta,$DateTimeFormat,2)
1617 If @error Then
1618 ; $Indx_CTime = "-"
1619 $Indx_CTime = "1601-01-01 00:00:00:000:0000"
1620 Else
1621 $Indx_CTime = $Indx_CTime & ":" & _FillZero(StringRight($Indx_CTime_tmp,4))
1622 EndIf
1623 $Indx_ATime = StringMid($Entry,$NextEntryOffset+64,16)
1624 $Indx_ATime = StringMid($Indx_ATime,15,2) & StringMid($Indx_ATime,13,2) & StringMid($Indx_ATime,11,2) & StringMid($Indx_ATime,9,2)
1625 $Indx_ATime_tmp = _WinTime_UTCTimeToLocalFileTime("0x" & $Indx_ATime)
1626 $Indx_ATime = _WinTime_UTCTimeFormat(Dec($Indx_ATime)-$tDelta,$DateTimeFormat,2)
1627 If @error Then
1628 ; $Indx_ATime = "-"
1629 $Indx_ATime = "1601-01-01 00:00:00:000:0000"
1630 Else
1631 $Indx_ATime = $Indx_ATime & ":" & _FillZero(StringRight($Indx_ATime_tmp,4))
1632 EndIf
1633 $Indx_MTime = StringMid($Entry,$NextEntryOffset+80,16)
1634 $Indx_MTime = StringMid($Indx_MTime,15,2) & StringMid($Indx_MTime,13,2) & StringMid($Indx_MTime,11,2) & StringMid($Indx_MTime,9,2)
1635 $Indx_MTime_tmp = _WinTime_UTCTimeToLocalFileTime("0x" & $Indx_MTime)
1636 $Indx_MTime = _WinTime_UTCTimeFormat(Dec($Indx_MTime)-$tDelta,$DateTimeFormat,2)
1637 If @error Then
1638 ; $Indx_MTime = "-"
1639 $Indx_MTime = "1601-01-01 00:00:00:000:0000"
1640 Else
1641 $Indx_MTime = $Indx_MTime & ":" & _FillZero(StringRight($Indx_MTime_tmp,4))
1642 EndIf
1643 $Indx_RTime = StringMid($Entry,$NextEntryOffset+96,16)
1644 $Indx_RTime = StringMid($Indx_RTime,15,2) & StringMid($Indx_RTime,13,2) & StringMid($Indx_RTime,11,2) & StringMid($Indx_RTime,9,2)
1645 $Indx_RTime_tmp = _WinTime_UTCTimeToLocalFileTime("0x" & $Indx_RTime)
1646 $Indx_RTime = _WinTime_UTCTimeFormat(Dec($Indx_RTime)-$tDelta,$DateTimeFormat,2)
1647 If @error Then
1648 ; $Indx_RTime = "-"
1649 $Indx_RTime = "1601-01-01 00:00:00:000:0000"
1650 Else
1651 $Indx_RTime = $Indx_RTime & ":" & _FillZero(StringRight($Indx_RTime_tmp,4))
1652 EndIf
1653 $Indx_AllocSize = StringMid($Entry,$NextEntryOffset+112,16)
1654 $Indx_AllocSize = Dec(StringMid($Indx_AllocSize,15,2) & StringMid($Indx_AllocSize,13,2) & StringMid($Indx_AllocSize,11,2) & StringMid($Indx_AllocSize,9,2))
1655 ; ConsoleWrite("$Indx_AllocSize = " & $Indx_AllocSize & @crlf)
1656 $Indx_RealSize = StringMid($Entry,$NextEntryOffset+128,16)
1657 $Indx_RealSize = Dec(StringMid($Indx_RealSize,15,2) & StringMid($Indx_RealSize,13,2) & StringMid($Indx_RealSize,11,2) & StringMid($Indx_RealSize,9,2))
1658 ; ConsoleWrite("$Indx_RealSize = " & $Indx_RealSize & @crlf)
1659 $Indx_File_Flags = StringMid($Entry,$NextEntryOffset+144,16)
1660 ; ConsoleWrite("$Indx_File_Flags = " & $Indx_File_Flags & @crlf)
1661 ; $Indx_File_Flags = StringMid($Indx_File_Flags,15,2) & StringMid($Indx_File_Flags,13,2) & StringMid($Indx_File_Flags,11,2) & StringMid($Indx_File_Flags,9,2)
1662 ; ConsoleWrite("$Indx_File_Flags = " & $Indx_File_Flags & @crlf)
1663 $Indx_File_Flags = StringMid(_SwapEndian($Indx_File_Flags),9,8)
1664 $Indx_File_Flags = _File_Attributes("0x" & $Indx_File_Flags)
1665 ; ConsoleWrite("$Indx_File_Flags = " & $Indx_File_Flags & @crlf)
1666 $Indx_NameLength = StringMid($Entry,$NextEntryOffset+160,2)
1667 $Indx_NameLength = Dec($Indx_NameLength)
1668 ; ConsoleWrite("$Indx_NameLength = " & $Indx_NameLength & @crlf)
1669 $Indx_NameSpace = StringMid($Entry,$NextEntryOffset+162,2)
1670 ; ConsoleWrite("$Indx_NameSpace = " & $Indx_NameSpace & @crlf)
1671 Select
1672 Case $Indx_NameSpace = "00" ;POSIX
1673 $Indx_NameSpace = "POSIX"
1674 Case $Indx_NameSpace = "01" ;WIN32
1675 $Indx_NameSpace = "WIN32"
1676 Case $Indx_NameSpace = "02" ;DOS
1677 $Indx_NameSpace = "DOS"

```

```

1678         Case $Indx_NameSpace = "03" ;DOS+WIN32
1679             $Indx_NameSpace = "DOS+WIN32"
1680     EndSelect
1681     $Indx_FileName = StringMid($Entry,$NextEntryOffset+164,$Indx_NameLength*2*2)
1682 ;     ConsoleWrite("$Indx_FileName = " & $Indx_FileName & @crlf)
1683     $Indx_FileName = _UnicodeHexToStr($Indx_FileName)
1684 ;
1685     $tmp0 = 0
1686     $tmp2 = 0
1687     $tmp3 = 0
1688     $tmp1 = 164+($Indx_NameLength*2*2)
1689     Do ; Calculate the length of the padding - 8 byte aligned
1690         $tmp2 = $tmp1/16
1691         If Not IsInt($tmp2) Then
1692             $tmp0 = 2
1693             $tmp1 += $tmp0
1694             $tmp3 += $tmp0
1695         EndIf
1696     Until IsInt($tmp2)
1697     $PaddingLength = $tmp3
1698 ;     ConsoleWrite("$PaddingLength = " & $PaddingLength & @crlf)
1699     $Padding = StringMid($Entry,$NextEntryOffset+164+($Indx_NameLength*2*2),$PaddingLength)
1700 ;     ConsoleWrite("$Padding = " & $Padding & @crlf)
1701     If $IndexFlags <> "0000" Then
1702         $SubNodeVCN = StringMid($Entry,$NextEntryOffset+164+($Indx_NameLength*2*2)+$PaddingLength,16)
1703         $SubNodeVCNLength = 16
1704     Else
1705         $SubNodeVCN = "-"
1706         $SubNodeVCNLength = 0
1707     EndIf
1708 ;     ConsoleWrite("$SubNodeVCN = " & $SubNodeVCN & @crlf)
1709     $NextEntryOffset = $NextEntryOffset+164+($Indx_NameLength*2*2)+$PaddingLength+$SubNodeVCNLength
1710     ReDim $IndxEntryNumberArr[1+$EntryCounter]
1711     ReDim $IndxMFTReferenceArr[1+$EntryCounter]
1712     ReDim $IndxMFTRefSeqNoArr[1+$EntryCounter]
1713     ReDim $IndxIndexFlagsArr[1+$EntryCounter]
1714     ReDim $IndxMFTReferenceOfParentArr[1+$EntryCounter]
1715     ReDim $IndxMFTParentRefSeqNoArr[1+$EntryCounter]
1716     ReDim $IndxCTimeArr[1+$EntryCounter]
1717     ReDim $IndxATimeArr[1+$EntryCounter]
1718     ReDim $IndxMTimeArr[1+$EntryCounter]
1719     ReDim $IndxRTimeArr[1+$EntryCounter]
1720     ReDim $IndxAllocSizeArr[1+$EntryCounter]
1721     ReDim $IndxRealSizeArr[1+$EntryCounter]
1722     ReDim $IndxFileFlagsArr[1+$EntryCounter]
1723     ReDim $IndxFileNameArr[1+$EntryCounter]
1724     ReDim $IndxNameSpaceArr[1+$EntryCounter]
1725     ReDim $IndxSubNodeVCNArr[1+$EntryCounter]
1726     $IndxEntryNumberArr[$EntryCounter] = $EntryCounter
1727     $IndxMFTReferenceArr[$EntryCounter] = $MFTReference
1728     $IndxMFTRefSeqNoArr[$EntryCounter] = $MFTReferenceSeqNo
1729     $IndxIndexFlagsArr[$EntryCounter] = $IndexFlags
1730     $IndxMFTReferenceOfParentArr[$EntryCounter] = $MFTReferenceOfParent
1731     $IndxMFTParentRefSeqNoArr[$EntryCounter] = $MFTReferenceOfParentSeqNo
1732     $IndxCTimeArr[$EntryCounter] = $Indx_CTime
1733     $IndxATimeArr[$EntryCounter] = $Indx_ATime
1734
1735     $IndxMTimeArr[$EntryCounter] = $Indx_MTime
1736     $IndxRTimeArr[$EntryCounter] = $Indx_RTime
1737     $IndxAllocSizeArr[$EntryCounter] = $Indx_AllocSize
1738     $IndxRealSizeArr[$EntryCounter] = $Indx_RealSize
1739     $IndxFileFlagsArr[$EntryCounter] = $Indx_File_Flags
1740     $IndxFileNameArr[$EntryCounter] = $Indx_FileName
1741     $IndxNameSpaceArr[$EntryCounter] = $Indx_NameSpace
1742     $IndxSubNodeVCNArr[$EntryCounter] = $SubNodeVCN
1743 ;     _ArrayDisplay($IndxFileNameArr,"$IndxFileNameArr")
1744     Until $NextEntryOffset+32 >= StringLen($Entry)
1745 EndFunc
1746
1747 Func _SetArrays()
1748     $IndxEntryNumberArr[0] = "Entry number"
1749     $IndxMFTReferenceArr[0] = "MFTReference"
1750     $IndxMFTRefSeqNoArr[0] = "MFTReference SeqNo"
1751     $IndxIndexFlagsArr[0] = "IndexFlags"

```

```

1751     $IndxMFTReferenceOfParentArr[0] = "Parent MFTReference"
1752     $IndxMFTParentRefSeqNoArr[0] = "Parent MFTReference SeqNo"
1753     $IndxCtimeArr[0] = "CTime"
1754     $IndxATimeArr[0] = "ATime"
1755     $IndxMTimeArr[0] = "MTime"
1756     $IndxRTimeArr[0] = "RTime"
1757     $IndxAllocSizeArr[0] = "AllocSize"
1758     $IndxRealSizeArr[0] = "RealSize"
1759     $IndxFileFlagsArr[0] = "File flags"
1760     $IndxFileNameArr[0] = "FileName"
1761     $IndxNameSpaceArr[0] = "NameSpace"
1762     $IndxSubNodeVCNArr[0] = "SubNodeVCN"
1763 EndFunc
1764
1765 Func _FillZero($inp)
1766     Local $inplen, $out, $tmp = ""
1767     $inplen = StringLen($inp)
1768     For $i = 1 To 4-$inplen
1769         $tmp &= "0"
1770     Next
1771     $out = $tmp & $inp
1772     Return $out
1773 EndFunc
1774
1775 ; start: by Ascend4nt -----
1776 Func _WinTime_GetUTCFileTimeToLocalFileTimeDelta()
1777     Local $iUTCFileTime=864000000000 ; exactly 24 hours from the origin (although 12 hours would be more
1778     $iLocalFileTime=_WinTime_UTCFileTimeToLocalFileTime($iUTCFileTime)
1779     If @error Then Return SetError(@error,@extended,-1)
1780     Return $iLocalFileTime-$iUTCFileTime ; /3600000000 = # hours delta (effectively giving the offset in hours from
1781 EndFunc
1782
1783 Func _WinTime_UTCFileTimeToLocalFileTime($iUTCFileTime)
1784
1785     If $iUTCFileTime<0 Then Return SetError(1,0,-1)
1786     Local $aRet=DllCall($COMMON_KERNEL32DLL,"bool","FileTimeToLocalFileTime",$iUTCFileTime,"uint64*",0)
1787     If @error Then Return SetError(2,@error,-1)
1788     If Not $aRet[0] Then Return SetError(3,0,-1)
1789     Return $aRet[2]
1790 EndFunc
1791
1792 Func _WinTime_UTCFileTimeFormat($iUTCFileTime,$iFormat=4,$iPrecision=0,$bAMPMConversion=False)
1793 ;~ If $iUTCFileTime<0 Then Return SetError(1,0,"") ; checked in below call
1794
1795 ; First convert file time (UTC-based file time) to 'Local file time'
1796 Local $iLocalFileTime=_WinTime_UTCFileTimeToLocalFileTime($iUTCFileTime)
1797 If @error Then Return SetError(@error,@extended,"")
1798 ; Rare occasion: a filetime near the origin (January 1, 1601!!) is used,
1799 ; causing a negative result (for some timezones). Return as invalid param.
1800 If $iLocalFileTime<0 Then Return SetError(1,0,"")
1801
1802 ; Then convert file time to a system time array & format & return it
1803 Local $vReturn=_WinTime_LocalFileTimeFormat($iLocalFileTime,$iFormat,$iPrecision,$bAMPMConversion)
1804 Return SetError(@error,@extended,$vReturn)
1805 EndFunc
1806
1807 Func _WinTime_LocalFileTimeFormat($iLocalFileTime,$iFormat=4,$iPrecision=0,$bAMPMConversion=False)
1808 ;~ If $iLocalFileTime<0 Then Return SetError(1,0,"") ; checked in below call
1809
1810 ; Convert file time to a system time array & return result
1811 Local $aSysTime=_WinTime_LocalFileTimeToSystemTime($iLocalFileTime)
1812 If @error Then Return SetError(@error,@extended,"")
1813
1814 ; Return only the SystemTime array?
1815 If $iFormat=0 Then Return $aSysTime
1816
1817 Local $vReturn=_WinTime_FormatTime($aSysTime[0],$aSysTime[1],$aSysTime[2],$aSysTime[3], _
1818     $aSysTime[4],$aSysTime[5],$aSysTime[6],$aSysTime[7],$iFormat,$iPrecision,$bAMPMConversion)
1819 Return SetError(@error,@extended,$vReturn)
1820 EndFunc
1821
1822 Func _WinTime_LocalFileTimeToSystemTime($iLocalFileTime)
1823     Local $aRet,$stSysTime,$aSysTime[8]=[-1,-1,-1,-1,-1,-1,-1,-1]
1824
1825     ; Negative values unacceptable

```

```

1825     If $iLocalFileTime<0 Then Return SetError(1,0,$aSysTime)
1826
1827     ; SYSTEMTIME structure [Year,Month,DayOfWeek,Day,Hour,Min,Sec,Milliseconds]
1828     $stSysTime=DllStructCreate("ushort[8]")
1829
1830     $aRet=DllCall($_COMMON_KERNEL32DLL, "bool", "FileTimeToSystemTime", "uint64*", $iLocalFileTime, "ptr", DllStructGetPtr($stSysTime), @error)
1831     If @error Then Return SetError(2,@error,$aSysTime)
1832     If Not $aRet[0] Then Return SetError(3,0,$aSysTime)
1833     Dim $aSysTime[8]=[DllStructGetData($stSysTime,1,1),DllStructGetData($stSysTime,1,2),DllStructGetData($stSysTime,1,4),
1834         DllStructGetData($stSysTime,1,6),DllStructGetData($stSysTime,1,7),DllStructGetData($stSysTime,1,8),DllStructGetData($stSysTime,1,9)]
1835     Return $aSysTime
1836 EndFunc
1837
1838 Func _WinTime_FormatTime($iYear,$iMonth,$iDay,$iHour,$iMin,$iSec,$iMilSec,$iDayOfWeek,$iFormat=4,$iPrecision=0,$bAMPMConversion=0)
1839     Local Static $_WT_aMonths[12]=["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"]
1840     Local Static $_WT_aDays[7]=["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]
1841
1842     If Not $iFormat Or $iMonth<1 Or $iMonth>12 Or $iDayOfWeek>6 Then Return SetError(1,0,"")
1843
1844     ; Pad MM,DD,HH,MM,SS,MSMSMSMS as necessary
1845     Local $sMM=StringRight(0&$iMonth,2),$sDD=StringRight(0&$iDay,2),$sMin=StringRight(0&$iMin,2)
1846     ; $sYY = $iYear ; (no padding)
1847     ; [technically Year can be 1-x chars - but this is generally used for 4-digit years. And SystemTime only goes to 2100]
1848     Local $sHH,$sSS,$sMS,$sAMPM
1849
1850     ; 'Extra precision 1': +SS (Seconds)
1851     If $iPrecision Then
1852         $sSS=StringRight(0&$iSec,2)
1853         ; 'Extra precision 2': +MSMSMSMS (Milliseconds)
1854         If $iPrecision>1 Then
1855             $sMS=StringRight('000'&$iMilSec,4)
1856             $sMS=StringRight('000'&$iMilSec,3);Fixed an erroneous 0 in front of the milliseconds
1857         Else
1858             $sMS=""
1859         EndIf
1860     Else
1861         $sSS=""
1862         $sMS=""
1863     EndIf
1864     If $bAMPMConversion Then
1865         If $iHour>11 Then
1866             $sAMPM=" PM"
1867             ; 12 PM will cause 12-12 to equal 0, so avoid the calculation:
1868             If $iHour=12 Then
1869                 $sHH="12"
1870             Else
1871                 $sHH=StringRight(0&($iHour-12),2)
1872             EndIf
1873         Else
1874             $sAMPM=" AM"
1875             If $iHour Then
1876                 $sHH=StringRight(0&$iHour,2)
1877             Else
1878                 ; 00 military = 12 AM
1879                 $sHH="12"
1880             EndIf
1881         EndIf
1882     Else
1883         $sAMPM=""
1884
1885         $sHH=StringRight(0 & $iHour,2)
1886     EndIf
1887     Local $sDateTimeStr,$aReturnArray[3]
1888
1889     ; Return an array? [formatted string + "Month" + "DayOfWeek"]
1890     If BitAND($iFormat,0x10) Then
1891         $aReturnArray[1]=$_WT_aMonths[$iMonth-1]
1892         If $iDayOfWeek>=0 Then
1893             $aReturnArray[2]=$_WT_aDays[$iDayOfWeek]
1894         Else
1895             $aReturnArray[2]=" "
1896         EndIf
1897         ; Strip the 'array' bit off (array[1] will now indicate if an array is to be returned)

```

```

1898     $iFormat=BitAND($iFormat,0xF)
1899 Else
1900     ; Signal to below that the array isn't to be returned
1901     $aReturnArray[1]=" "
1902 EndIf
1903
1904 ; Prefix with "DayOfWeek " ?
1905 If BitAND($iFormat,8) Then
1906     If $iDayOfWeek<0 Then Return SetError(1,0,"") ; invalid
1907     $sDateTimeStr=$_WT_aDays[$iDayOfWeek]& ', '
1908     ; Strip the 'DayOfWeek' bit off
1909     $iFormat=BitAND($iFormat,0x7)
1910 Else
1911     $sDateTimeStr=" "
1912 EndIf
1913
1914 If $iFormat<2 Then
1915     ; Basic String format: YYYYMMDDHHMM[SS[MSMSMS[ AM/PM]]]
1916     $sDateTimeStr&=$iYear&$sMM&$sDD&$sHH&$sMin&$sSS&$sMS&$sAMPM
1917 Else
1918     ; one of 4 formats which ends with " HH:MM[:SS[:MSMSMS[ AM/PM]]]"
1919     Switch $iFormat
1920         ; /, : Format - MM/DD/YYYY
1921         Case 2
1922             $sDateTimeStr&=$sMM&'/'&$sDD&'/'
1923         ; /, : alt. Format - DD/MM/YYYY
1924         Case 3
1925             $sDateTimeStr&=$sDD&'/'&$sMM&'/'
1926         ; "Month DD, YYYY" format
1927         Case 4
1928             $sDateTimeStr&=$_WT_aMonths[$iMonth-1]&' '&$sDD&', '
1929         ; "DD Month YYYY" format
1930         Case 5
1931             $sDateTimeStr&=$sDD&' '&$_WT_aMonths[$iMonth-1]&' '
1932         Case 6
1933             $sDateTimeStr&=$iYear&'-'&$sMM&'-'&$sDD
1934
1935             $iYear=''
1936         Case Else
1937             Return SetError(1,0,"")
1938     EndSwitch
1939     $sDateTimeStr&=$iYear&' '&$sHH&' ':'&$sMin
1940     If $iPrecision Then
1941         $sDateTimeStr&=: '&$sSS
1942         If $iPrecision>1 Then $sDateTimeStr&=: '&$sMS
1943     EndIf
1944     $sDateTimeStr&=$sAMPM
1945 EndIf
1946 If $aReturnArray[1]<>" " Then
1947     $aReturnArray[0]=$sDateTimeStr
1948     Return $aReturnArray
1949 EndIf
1950 Return $sDateTimeStr
1951 EndFunc
1952
1953 ; end: by Ascend4nt -----
1954
1955 Func _DecodeNameQ($NameQ)
1956     For $name = 1 To UBound($NameQ) - 1
1957         $NameString = $NameQ[$name]
1958         If $NameString = "" Then ContinueLoop
1959         $FN_AllocSize = Dec(_SwapEndian(StringMid($NameString,129,16)),2)
1960         $FN_RealSize = Dec(_SwapEndian(StringMid($NameString,145,16)),2)
1961         $FN_NameLength = Dec(StringMid($NameString,177,2))
1962         $FN_NameSpace = StringMid($NameString,179,2)
1963         Select
1964             Case $FN_NameSpace = '00'
1965                 $FN_NameSpace = 'POSIX'
1966             Case $FN_NameSpace = '01'
1967                 $FN_NameSpace = 'WIN32'
1968             Case $FN_NameSpace = '02'
1969                 $FN_NameSpace = 'DOS'
1970             Case $FN_NameSpace = '03'
1971                 $FN_NameSpace = 'DOS+WIN32'
1972             Case Else
1973                 $FN_NameSpace = 'UNKNOWN'

```

```
1972         EndSelect
1973     $FN_FileName = StringMid($NameString,181,$FN_NameLength*4)
1974     $FN_FileName = _UnicodeHexToStr($FN_FileName)
1975     If StringLen($FN_FileName) <> $FN_NameLength Then $INVALID_FILENAME = 1
1976     Next
1977     Return
1978 EndFunc

1979
1980 Func _Usage()
1981     ConsoleWrite("Usage:" & @CRLF)
1982     ConsoleWrite("RawDir.exe mode path" & @CRLF)
1983     ConsoleWrite(" mode can be 1 or 2. 1 is verbose output. 2 is more compact output." & @CRLF)
1984
1985     ConsoleWrite(" path is the path to perform directory listing on." & @CRLF)
1986     ConsoleWrite("Example printing verbose output on the path C:\tmp" & @CRLF)
1987     ConsoleWrite(" RawDir.exe 1 C:\tmp" & @CRLF)
1988     ConsoleWrite("Example printing compact output on the root of the C: volume" & @CRLF)
1989     ConsoleWrite(" RawDir.exe 2 C:\" & @CRLF)
1990 EndFunc
1991
1992 Func _AlignString($input,$length)
1993     While 1
1994         If StringLen($input)=$length Then ExitLoop
1995         $input = " "&$input
1996     WEnd
1997     Return $input
1998 EndFunc
```

