EVERYTHING YOU WANTED TO KNOW ABOUT EXTENDED ATTRIBUTES

WELL ALMOST

EXTENDED ATTRIBUTES

- Extended attributes form an important part of OS/2 - eCS. In this presentation:
  - what EA's are
  - relationship to the file system
  - types
  - uses
  - pitfalls
  - standard and common EA's
  - EA utilities
  - API's
  - structure
  - some references on how to program EA's

ATTRIBUTES

- In OS/2 and other systems we have attributes and extended attributes

- The standard attributes are:
  - system
  - read only
  - archive
  - hidden

Extended attributes

- Extended attributes are:

  file system features that enables users to associate computer files with metadata not interpreted by the file system, whereas regular attributes have a purpose strictly defined by the file system (such as permissions or records of creation and modification times).

  NOTE : A DIRECTORY IS ALSO A FILE IN THIS CONTEXT AND MAY HAVE EA'S

Extended attributes

- IN OS/2 The number of EA’s per file is not limited
- But the total size of all EA’s associated with one file is limited to 64K!

Extended attributes

- Extended attributes are available in:
  - AIX
  - Linux (if enabled in the kernel)
  - OS/2
    - FAT indirectly using the file "EA DATA. SF“
    - FAT32 (if /EAS is used in the driver!)
    - JFS
    - HPFS
  - FreeBSD
  - OS/X
  - Solaris
  - Windows

Extended attributes non FAT

- In both HPFS and JFS the EA is part of the file itself.

- In HPFS for example if the EAs associated with a given file or directory are small enough, they will be stored right in the Fnode. If the total size of the EAs is too large, they are stored outside the Fnode in sector runs, and a B+ Tree of allocation sectors can be created to describe the runs. If a single EA gets too large, it can be pushed outside the Fnode into a B+ Tree of its own.

Extended attributes in FAT

- Fat Directory Entry is 32 bytes and has the following structure:
  - Filename: 8 bytes (0 – 7)
  - Extension: 3 bytes (8 – 10)
  - Attributes: 1 byte (11)
  - Reserved: 10 bytes (12-21)
  - Time 2 bytes (22-23)
  - Date 2 bytes (24-25)
  - Fat cluster 2 bytes (26-27)
  - Size 4 bytes (28-31)

Extended attributes FAT

- In os2 the FAT directory reserved entry (bytes 12-21) is used to point to the EA data.
- The last two bytes (20 & 21) point to the clusters containing the EA .
- The file "EA DATA. SF“ is used to hold or own the allocation of the EA data

Extended attributes FAT32

- In os2 the FAT directory reserved entry (bytes 12-21) is used to indicate that there is EA data.
- The EA is contained in an additional file with the same name as the original file but with the suffix :

  “ EA. SF” **(SPACE BEFORE THE ‘E’ AND AFTER THE ‘.’)**

  Example “file.bmp” has EA’s in “file.bmp EA. SF”
- This file is hidden by the OS/2 driver and cannot be seen. It is however visible when using other operating systems!
- The file has a similar format as a file created with EAUTIL

Why Extended attributes

- Additional information about a file is handy:

  Examples:
  The icon of an executable is in an EA
  The long filename
  The codepage used to encode the data
  The version number of a file
  etc

Extended attributes OS/2

- There are two major types of EA’s in OS/2
  - Critical (I have not yet found a critical EA!)
  - Non critical (Program will still work i.e. missing icon)

Extended attributes OS/2

- Each EA’s has an identification or type tag.
- This is done to indicate the format of the EA and the data and to enable correct decoding of the information.
- Some EA’s contain EA’s within an EA!

Extended Attribute Types

There are a number of predefined EA types

EAT_BINARY	FFFE	Binary data
EAT_ASCII	FFFD	ASCII text
EAT_BITMAP	FFFB	Bit map data
EAT_METAFILE	FFFA	Metafile data
EAT_ICON	FFF9	Icon data
EAT_EA	FFEE	ASCII name of another EA associated with the file
EAT_MVMT	FFDF	Multi-Valued, Multi-Typed data
EAT_MVST	FFDE	Multi-Valued, Single-Typed data
EAT_ASN1	FFDD	ASN.1 field data; an ISO EA standard

EA Names

- An EA name is:
  - either for a Standard Extended Attribute (SEA) which is defined by the system and always starts with a . (dot)
  - Free format

SEA´s

All Standard Extended Attributes start with a dot ´.´

The SEA´s are:

.APPTYPE	.ASSOCTABLE
.CHECKSUM	.CLASSINFO
.CODEPAGE	.COMMENTS*
.EXPAND	.HISTORY
.ICON*	.ICON1*
.ICONPOS	.KEYPHRASES*
.LONGNAME	.MMPREF_MMIMAGE
.PREVNAME	.SUBJECT*
.TYPE	.VERSION

* EA MANIPULATION VIA THE WPS

Some common non SEA EA's

FONT_INFO
MMBPS
MMNUMAUDIOBYTES
MMPLAYTIMEMS
OPTIONALDRIVERFILES
REXX.LITERALPOOL
REXX.TOKENSIMAGE
VENDORNAME
MMBITRATE
MMCHANNELS
MMPLAYTIME
MMSAMPLERATE
REQUIREDDRIVERFILES
REXX.METACONTROL
REXX.VARIABLEBUF

Standard Extend Attributes Formats

.ASSOCTABLE

The .ASSOCTABLE extended attribute (EA) contains information that associates data files with the applications that create them or that know how to use them. The .ASSOCTABLE extended attribute enables an application to indicate the type, extension, and icon for the data files it recognizes. The .ASSOCTABLE EA also contains an ownership flag. This tells OS/2 which application to run when the user double-clicks the mouse on a given data file.

Because programs can understand and reference data files generated by other programs, this EA can be used to link a program with those files.

The name of this EA consists of the string ".ASSOCTABLE". The value of this EA contains application information and consists of multi-valued, multi-typed fields that link the application with:

the file type (that is, the value of a .TYPE EA), the file extension, and icon data for data files that it generates or references.

The .ASSOCTABLE EA associates icons by file-type. The data file's file-type is indicated in the .TYPE EA, or, if the data file does not have a .TYPE EA, by the extension.

This data can be installed automatically by OS/2.

The format of the EA is as follows.

```
EAT_MVMT 0000 0004 EAT_ASCII  .TYPE name
                   EAT_ASCII   file extension
                   EAT_BINARY  flags
                   EAT_ICON    icon data
 .
END
```

Standard Extend Attributes Formats

.ASSOCTABLE

The association_name is the name of a file type that the Resource Compiler understands. (This is the same name found in the .TYPE field of data files.)

The extension is the three letter file extension that is used to identify files of this type, if they have no .TYPE EA entry. (Three letter extensions should be used so that FAT file systems can make use of this EA). This field can be empty.

The icon_filename is the name of the file that contains the icon that is to be used to represent this file type. (This field can also be empty.)

The .ASSOCTABLE flag indicates that the program is the default application for data files with the specified type. This determines the program OS/2 will start when the file is double-clicked with the mouse.

If more than one program has marked itself as the EAF_DEFAULTOWNER for a particular data file .TYPE, OS/2 will not know which program to run when the file of this .TYPE is double-clicked on with the mouse. If no program is marked as the EAF_DEFAULTOWNER for a particular data file .TYPE, OS/2 will be similarly confused. In both cases, OS/2 provides the user with a list of applications that understand the file .TYPE, regardless of whether the application is the owner or not. The user selects the program to run from this list.

The flag entry indicates whether the application owns the file or merely recognizes the .TYPE. If this flag is set, the entry describing data files of this type cannot be edited. This flag is specified if a previously defined icon in the ASSOCTABLE is to be reused. Entries with this flag set have no icon data defined. The icon used for this entry will be the icon used for the previous entry.

EAF_ flags can be ORed together when specified in the ASSOCTABLE. The EAF_ flags are defined in PMWIN.H and PMWIN.INC.

Standard Extend Attributes Formats

.ASSOCTABLE

.ASSOCTABLE Example

For example, My_Company's application, My_Application, generates or references data files that have the following .TYPE names:

```
My_Company My_Application documentation
My_Company My_Application macros
My_Company My_Application spreadsheet
My_Company My_Application chart
Your_Company Your_Application forecast
```

The source for the .ASSOCTABLE extended attribute in the resource file for My_Application could look like the following.

```
ASSOCTABLE
BEGIN
"My_Company My_Application documentation", "DOC", EAF_DEFAULTOWNER, My_App.ICO
"My_Company My_Application macros", "MAC", EAF_DEFAULTOWNER+EAF_REUSEICON
"My_Company My_Application spreadsheet", "SPR", EAF_DEFAULTOWNER+EAF_REUSEICON
"My_Company My_Application chart", "CHT", EAF_DEFAULTOWNER+EAF_REUSEICON
"Your_Company Your_Application forecast", "FOR", 0
END
```

Standard Extend Attributes Formats

.ASSOCTABLE

My_Application can load and use some files generated by Your_Application. However, because My_Application is not the default owner of those files, OS/2 does not run My_Application when the user double-clicks on the files with the mouse.

The following example illustrates how the value of the .ASSOCTABLE EA for My_Application might look. It is a multi-valued, multi-typed EA with five multi-valued, multi-typed entries (one for each file type referenced or generated by the application).

```
EAT_MVMT   0000 0005          ; There are 5 associated file types

EAT_MVMT   0000 0004          ; Description of 1st associated file type
EAT_ASCII  0027 My_Company My_Application documentation  ; File type
EAT_ASCII  0003 DOC           ; File extension
EAT_BINARY flags              ; Flags
EAT_ASCII  icon data                  ; Physical icon data

EAT_MVMT   0000 0004          ; Description of 2nd associated file type
EAT_ASCII  0020               My_Company My_Application macros
EAT_ASCII  0003 MAC
EAT_BINARY flags
EAT_ICON   icon data
```

Standard Extend Attributes Formats

.CLASSINFO, .CODEPAGE, .COMMENTS

.CLASSINFO

The .CLASSINFO extended attribute saves the instance data for a file system object.

.CODEPAGE

The .CODEPAGE extended attribute (EA) contains the code page for the file. If this extended attribute is not provided, the code page of the file is the system default or is defined by the application.

The code page of the EA data associated with the file is assumed to be that of the file, unless the EA entry is specifically overridden in the code page field in the multi-valued extended attribute data type.

.COMMENTS

The .COMMENTS extended attribute (EA) contains miscellaneous notes or reminders about the file (for example, peculiarities, restrictions, or requirements).

The name of this EA consists of the string ".COMMENT". The value of this EA consists of miscellaneous notes and can be multi-valued and of any type.

Standard Extend Attributes Formats

.HISTORY

The .HISTORY extended attribute (EA) contains the modification history for a file object, indicating the author of the file and all subsequent changes. Each entry is separate field in a multi-value field and consists of be ASCII characters only.

The name of this EA consists of the string ".HISTORY". The value of this EA contains the modification history for a file object and can be multi-valued, with each action entry described in a separate field.

Each entry in the .HISTORY field has the following format:

```
PERSON  ACTION(created, changed or printed)  DATE
```

For example, the following .HISTORY extended attribute contains two entries:

```
EAT_MVMT 0000 0002
        EAT_ASCII 0017 Joe          Created  2/10/88
        EAT_ASCII 0017 Harry        Changed  2/11/88
```

This extended attribute can potentially become quite large. To avoid unwanted growth, an application can let the user decide when an entry should be added to this extended attribute. For example, there are some cases when it is important to note when a document is printed. However, it is probably unnecessary to note it every time the file is printed.

Standard Extend Attributes Formats

.ICON

The .ICON extended attribute (EA) specifies the icon to be used for the file representation, for example when the application is minimized. This extended attribute contains the physical icon data used to represent the file object.

If there is no .ICON EA, OS/2 can use the .TYPE entry to determine a default icon to use for the particular file. If there is an .ICON entry, however, it is used instead of the default icon.

The name of this EA consists of the string ".ICON". The value of this EA contains the physical icon data and has the following format:

EAT_ICON	data_length	data
WORD	DWORD	

The data is of type BITMAPARRAYFILEHEADER and is used to specify an array of one device-dependent and one device-independent icon bit maps. The GpiLoadBitmap and WinLoadPointer functions support this icon file format.

It is best to provide as much icon information as possible. Ideally, an icon should be 64-by-64 bits in 8-color, device-independent format.

Standard Extend Attributes Formats

.ICON

The Icon Editor is used to create the icon, which is saved in an icon file. The .ICON extended attribute for an application is created by the Resource Compiler as part of the compile process by specifying the DEFAULTICON keyword, as in:

DEFAULTICON <filename.ico>

This keyword uses the icon definition contained in the specified icon file (FILENAME.ICO) to create the .ICON EA for the application.

Applications store the binary icon data in this extended attribute. To install icons for data files, the applications can use the .ASSOCTABLE extended attribute, or DosSetPathInfo.

Standard Extend Attributes Formats

.ICON1, ICONPOS

.ICON1

The .ICON1 extended attribute stores the animation icon (for example, the open folder icon) for a folder.

.ICONPOS

The .ICONPOS extended attribute saves the icon positioning information for a folder.

Standard Extend Attributes Formats

.KEYPHRASES

The .KEYPHRASES extended attribute (EA) contains key text phrases for the file. Such phrases can be used in performing a database-type search or in helping the user understand the nature of the file.

The name of this EA consists of the string ".KEYPHRASES". The value of this EA consists of key phrases in ASCII.

Key phrases are represented as ASCII characters. Multiple key phrases can be stored in the value of this extended attribute, each stored in a separate entry in a multi-valued field.

For example, the following extended attribute contains three key phrases:

```
EAT_MVST 0000 0003 EAT_ASCII 0008 ABC Inc.
                   EAT_ASCII 000A Salesman A
                   EAT_ASCII 000F Product X sales
```

If there is more than one key phrase, each should be stored in a separate entry in a multi-value field.

Standard Extend Attributes Formats

.PREVNAME, .SUBJECT

.PREVNAME

The .PREVNAME extended attribute saves the old class name when the user requests that an object, which is a descendent of WPDataFile, becomes another subclass of WPDataFile.

.SUBJECT

The .SUBJECT extended attribute (EA) contains a brief summary of the content or purpose of the file object it is associated with.

The name of this EA consists of the string ".SUBJECT". The value of this EA consists of a single-valued ASCII string that contains the purpose of the file object.

The length of this field must be less than 40 character

Standard Extend Attributes Formats

.TYPE

The .TYPE extended attribute (EA) indicates the file-type of the file object it is associated with. It is similar to a file name extension.

The name of this EA consists of the string ".TYPE". The value of this EA contains the file object's file-type. The following file types are predefined:

Plain text	
OS/2 command file	DOS command file
Executable	Metafile
Bit map	Icon
Binary data	Dynamic link library
C code	Pascal code
BASIC code	COBOL code
FORTRAN code	Assembler code
Library	Resource file
Object code	

Data files only require identification of the file type. For data files without EAs, the file type is derived from the file extension, if there is one.

Standard Extend Attributes Formats

.TYPE

File object types are represented as length-preceded ASCII strings, uniquely identifying the file object's type. This identifier is referenced within the application's .ASSOCTABLE EA in order to bind the data file type to the application. It is important that this name be a unique identifier because all file type names are public data. For example, if application A and application B both had a type name of SPREADSHEET, the filing system would not be able to identify A's SPREADSHEET from B's SPREADSHEET.

The recommended convention for defining file object types is:

Company_name
Application_name
Application-specific_name

For example, spreadsheet files generated by My_Application written by My_Company might have a file object type of the following.

    My_Company My_Application Spreadsheet

Type names must be ASCII characters and case is significant.

Note: The performance of extended attributes is dependent on the file system. Because some file systems store extended attributes in first-in/first-out (FIFO) order, it is important to write the .TYPE entry first so OS/2 can access that information quickly.

Standard Extend Attributes Formats

.VERSION

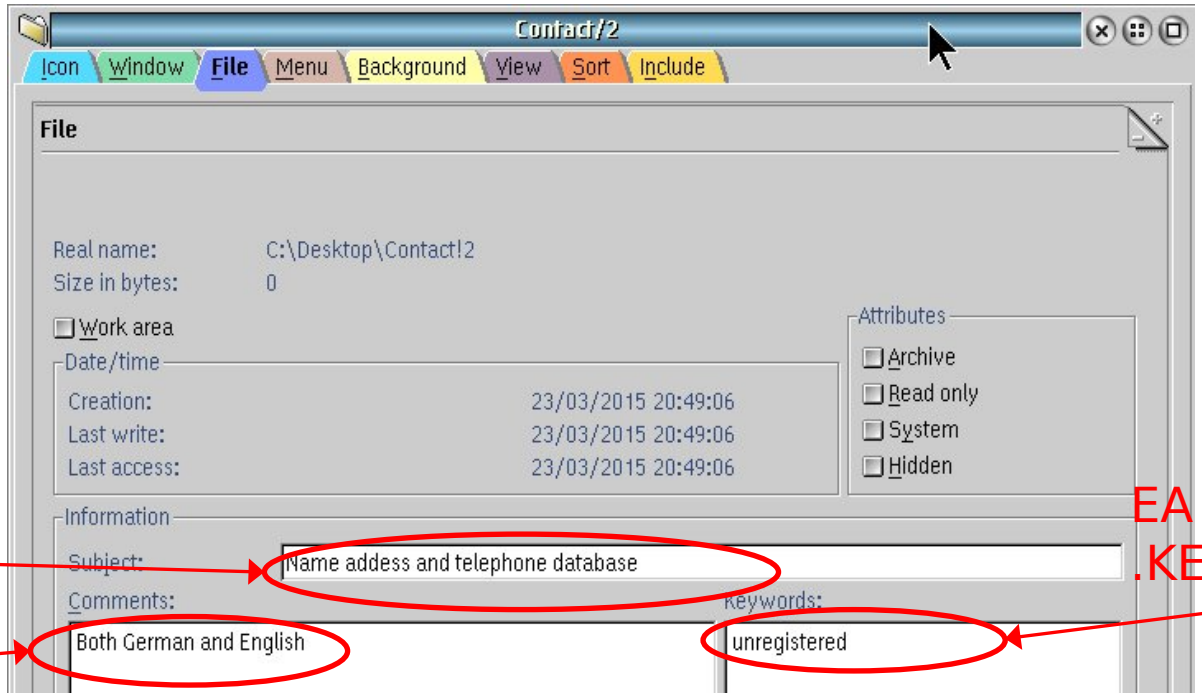
The .VERSION extended attribute (EA) contains the version number of the file format, as shown below.

    My_Application 1.0

The name of this EA consists of the string ".VERSION". The value of this EA contains the file object version number. This attribute can be ASCII or binary. Only the application that created the file object should modify the value of this EA. It can also be used to indicate an application or dynamic link library version number.

What can you do with EA's

- Certain EA's are directly available to you and can be manipulated via the Workplace Shell :-



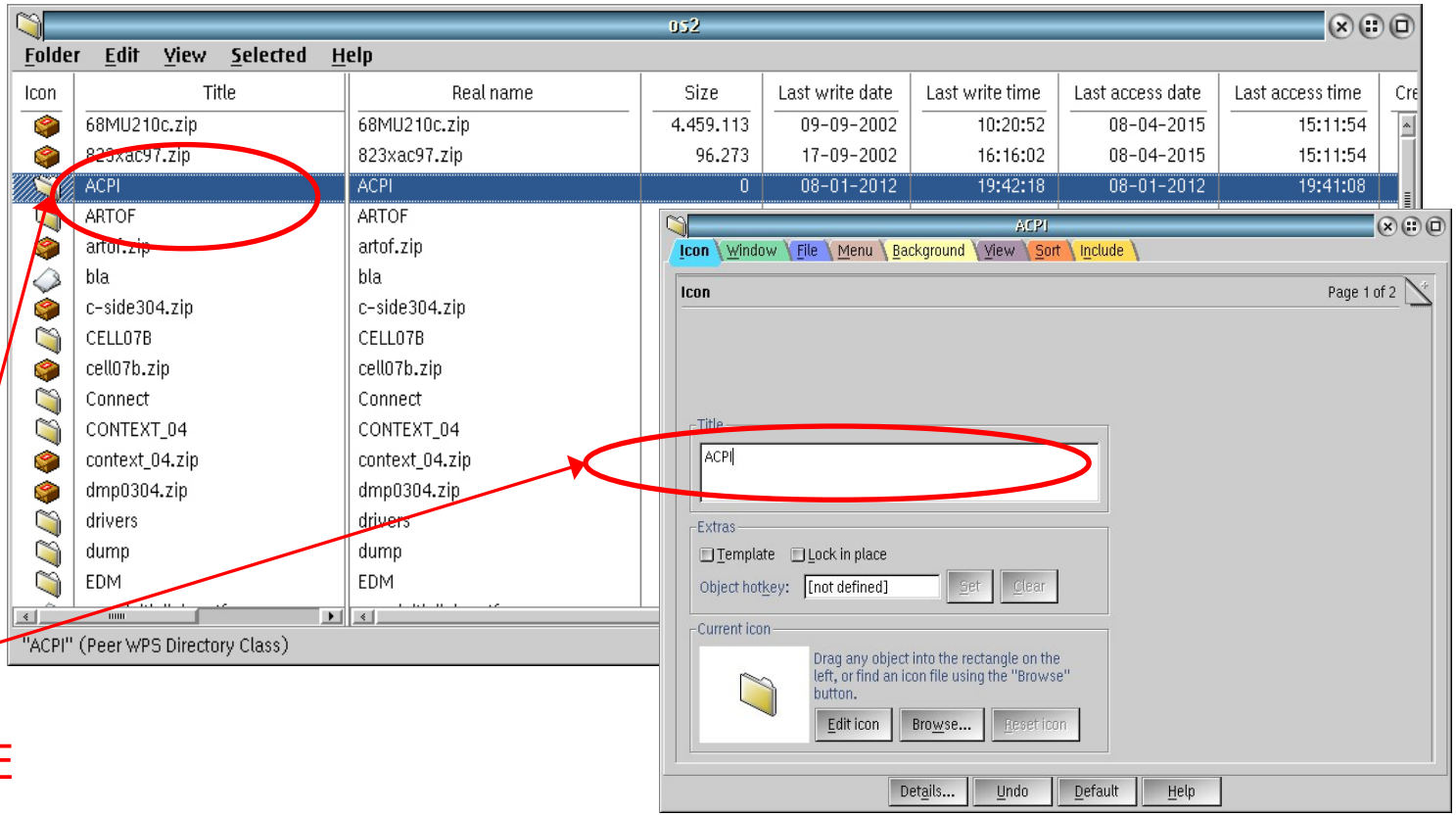
EA = .SUBJECT

EA = .KEYPHRASES

EA = .COMMENTS

What can you do with EA's

- Certain EA's are directly available to you and can be manipulated via the Workplace Shell



EA = .LONGNAME

Copying EA’s

When copying files, EA’s are automatically copied using either the WPS or the ‘copy’ command from the command line or any EA aware program

But only when the destination file system supports EA’s!

If a file has a .LONGNAME EA is edited and then saved using the SaveAs option then in most cases the value of this EA is not updated to reflect the new name!

Copying EA's

If the destination does not support EA's then:

–The workplace shell will not give an error!

–The command line copy command may give an error if the /F option is used

–4os2 copy command does not have the /F option

The only true check is to write and then read an EA to see if its there!

EA utility Programs

EABROWSE

EABROWSE - Henk Kelder (part of WPTOOLS)

–Displays all sorts of EA’s on a file per file basis together with metadata

–Cannot modify data

–Can delete an EA

EA utility Programs

SearchPlus

SearchPlus – Keith Merrington

–Can find files with:

- EA’s
- specific EA’s
- Specific data in a (specific) EA

EA utility Programs

LongNameCheck

LongNameCheck – Keith Merrington

–Can find & select files with duplicate information in the .LONGNAME attribute (per directory)

–Can identify possible incorrect .LONGNAME information

–Replace .LONGNAME data with real name

- Edit .LONGNAME name
- Restore .LONGNAME data

EA utility Programs

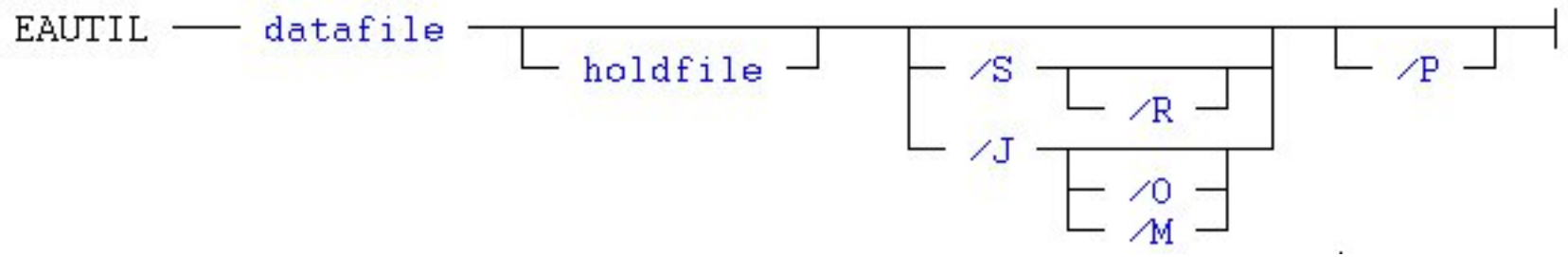
EA Viewer

EA Viewer – M Kimes (part of FM/2)

- Displays all sorts of EA's on a file per file basis together with metadata
- Can modify EA data for ASCII EA types
- Can Add ASCII and MultiValue ASCII EA's
- Can Delete any EA

EA utility Programs

EAUtil



Can split and join EA´s from a file

/S split /R replace

/J join /O overwrite /M merge

/P (preserve)

EAutil output file structure

Bytes	Description
0 - 3	Size of file
4 - 5	Length EA Name
6 - 7	Size of EA
8 - n	EA Name
n+1	EA DATA where bytes:
	0 - 1 EAType
	2 Critical / noncritical FLAG
	3 Size of EA
x	Length EA Name
x+2	Size of next EA
x+4	EA Name
Etc.	

EA Facts

There is no way to see if an EA has been changed or deleted except by knowing what EA’s were previously associated with a file or directory and if the size of the EA has changed

EA’s - The unwritten rule

When an EA is changed/added using the WPI the last write date is unchanged!

When an EA is changed/added using the WPI the archive bit is unchanged!

A programmer must take care of this when writing an EA as the system will set the archive bit and update the last write date!

EA’s & the relevant API’s

The following API’s are available with regard to Extended attributes:

DosFindFirst	-	Get size of EA’s or read EA & data,specified EA(s)
DosFindNext		“ “
DosQueryFileInfo	-	Get size of EA’s or read EA & data, specified EA(s)
DosQueryPathInfo		“ “
DosEnumAttributes	-	Enumerate EA’s (read EA’s and data)
DosOpen	-	Write EA’s & data
DosSetFileInfo	-	Write Extended Attributes
DosSetPathInfo	-	Write Extended Attributes

Since DosQueryFileInfo & DosSetFileInfo both require a handle to a file they cannot be used for EA’s of a directory as it is not possible to obtain a handle for a directory!

EA’s & the relevant API’s

DosFindFirst & DosFindNext

ulrc = DosFindFirst(pszFileSpec, phdir, flAttribute, pfindbuf, cbBuf, pcFileNames, ulInfoLevel);

The level of file info returned depends on ulInfoLevel

FIL_STANDARD - file size and attribute info
FILEFINDBUF3 data structure

FIL_QUERYEASIZE - The size of all EA’s for this file
FILEFINDBUF4 data structure

The buffer required to hold the entire EA set is less than or equal to twice the size of the EA size!

FIL_QUERYEASFROMLIST - EA’s as specified in EAOP2 structure

EA’s & the relevant API’s

DosQueryFileInfo & DosQueryPathInfo

ulrc = DosQueryPathInfo(pszPathName, ulInfoLevel, pInfoBuf, cbInfoBuf);

The level of file info returned depends on ulInfoLevel

FIL_STANDARD - file size and attribute info
FILESTATUS3 data structure

FIL_QUERYEASIZE - The size of all EA’s for this file
FILESTATUS4 data structure

The buffer required to hold the entire EA set is less than or equal to twice the size of the EA size!

FIL_QUERYEASFROMLIST - EA’s as specified in EAOP2 structure

EA’s & the relevant API’s

DosEnumAttributes

```
ulrc =      DosEnumAttribute(ulRefType, pvFile, ulEntry,
pvBuf,
            cbBuf, pulCount, ulInfoLevel);
```

The level of file info returned depends on ulInfoLevel BUT only one level is available!

Data returned (pvBuf) is in the FEA2 format

EA’s & the relevant API’s

DosOpen

ulrc = DosOpen (pszFileName, pHf, pulAction, cbFile,
ulAttribute,fsOpenFlags, fsOpenMode,
peaop2)

EA’s to be written must be contained in a structure of the type FEA2 as indicated by peaop2.

This is only used when opening a new file, truncating or replacing an existing file!

EA’s & the relevant API’s

DosSetFileInfo & DosSetPathInfo

ulrc = DosSetPathInfo(pszPathName, ulInfoLevel, pInfoBuf, cbInfoBuf, flOptions);

The level of file info set depends on ulInfoLevel

FIL_STANDARD - file size and attribute info FILEFINDBUF3 data structure

FIL_QUERYEASIZE - The EA’s to be written in presented in a EAOP2 data structure

EA’s & the relevant API’s

Structures

FILEFINDBUF3, FILEFINDBUF4 and FILESTATUS3 , FILESTATUS4 data structures

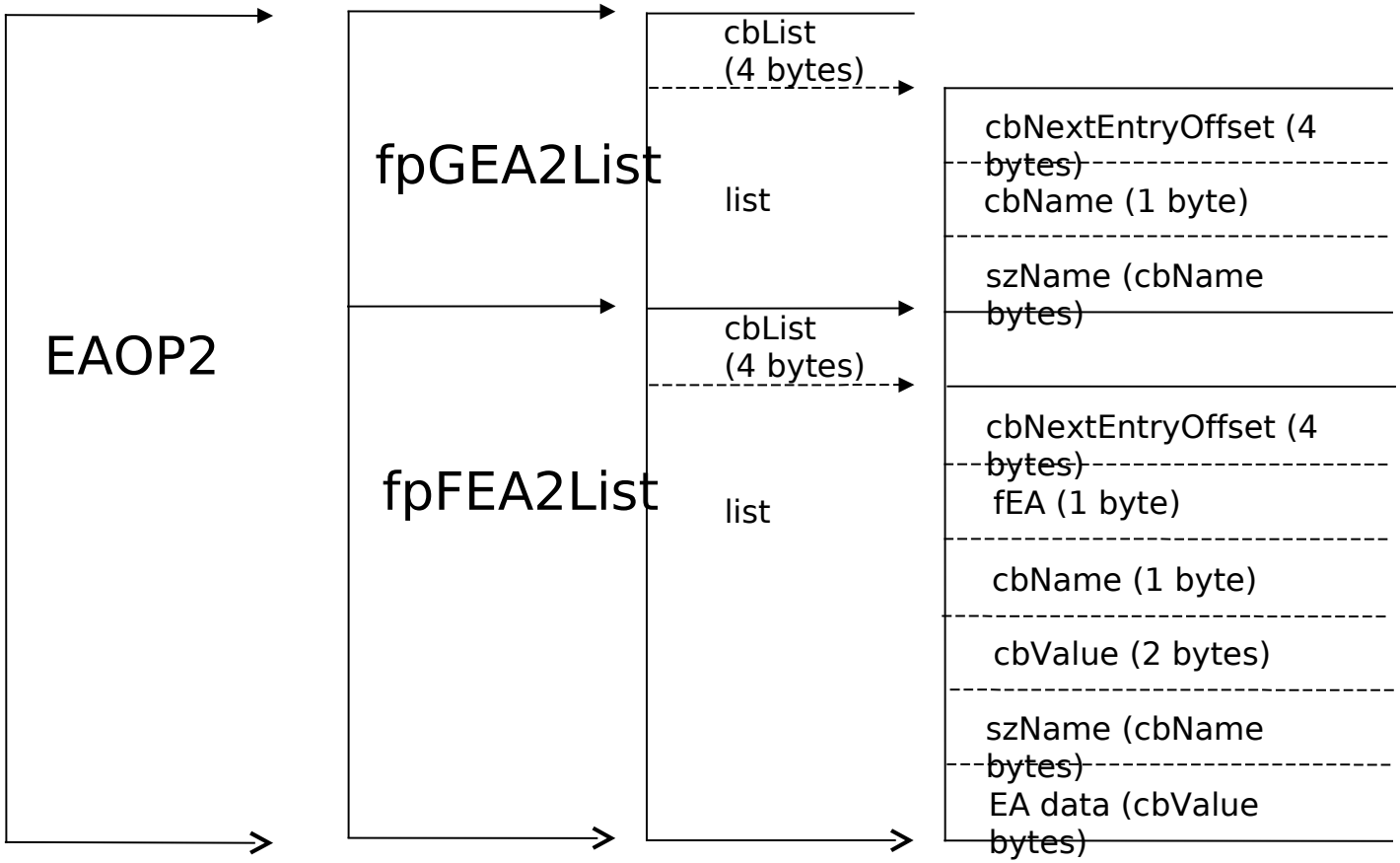
```
ULONG   oNextEntryOffset;       Offset of next entry.
FDATE   fdateCreation;          Date of file creation.
FTIME   ftimeCreation;          Time of file creation.
FDATE   fdateLastAccess;        Date of last access.
FTIME   ftimeLastAccess;        Time of last access.
FDATE   fdateLastWrite;         Date of last write.
FTIME   ftimeLastWrite;         Time of last write.
ULONG   cbFile;                 Size of file.
ULONG   cbFileAlloc;            Allocated size.
ULONG   attrFile;               File attributes.
ULONG   cbList;                 Size of the file's extended attributes.
UCHAR   cchName;                Length of file name.
CHAR    achName[CCHMAXPATHCOMP];  File name including null terminator.
```

*Only in FILEFINDBUF3 and FILEFINDBUF4 data structure

*Only in FILEFINDBUF4 and FILESTATUS4 data structure

EA’s & the relevant API’s

Structures



Reading an EA

The procedure for reading an EA is:

–Get size of EA's in a file/directory using the parameter FIL_QUERYEASIZE(L)

(use DosFindFirst, DosFindNext, or DosQueryFileInfo/DosQueryPathInfo)

–Reserve memory

–Enumerate the EA's in a loop using DosEnumAttribute

Writing an EA

The procedure for writing an EA is:

–Reserve memory and create the EA structures taking care that each list entry is on a double-word boundary

–Fill the structure with the EA information

–Write EA’s using DosOpen or DosSetFileInfo /DosSetPathInfo

EA Programming information

The Art of OS/2 Programming

(http://www.laser.ru/evgen/articles/ARTofOS2/aos2p_4.html)

Encapsulating Extended Attributes - Part 1 & 2

http://www.edm2.com/0404/eas1.html

Extended Attributes - what are they and how can you use them

*www.howzatt.demon.co.uk/articles/06**may93.html***

THANK YOU

Any Questions