Mesa 2 version 2.2
File Format Documentation

Mesa 2 version 2.2 saves its information in tokenized workbook files.

File versions:
There are several versions of these files:

1) Mesa 2 version 2.0.x (Mesa2DPP)
2) Mesa 2 version 2.1.x (Mesa DPP)
3) Mesa 2 version 2.2 (Mesa JDK)
4) Mesa 2 version 2.2 compressed (Mesa CMP)
   (minimal and maximal compression)

The purpose of this document is to describe the third format, the normal Mesa 2 version 2.2 uncompressed file. The other formats may be documented at a future date.

To determine which type of file it is, you need to look at the first 8 bytes of the file. It should be one of the 4 types listed above. For example, by default, Mesa 2 saves files in 2.2 uncompressed format. The first 8 bytes of those files will be "Mesa JDK". (ASCII text). Mesa 2 version 2.2 can also generate "Mesa DPP" and "Mesa CMP" files as well as read all 4 types of files. It cannot generate "Mesa2DPP" files. If you have people using a 2.0.x version of Mesa 2, we strongly advise they upgrade to version 2.2 (minimal cost) or at least to 2.1.6 (free upgrade).

Endian:
At this time, all numbers in all versions of Mesa 2 files are stored in the Intel "little endian" notation. The high order bytes will appear AFTER the low order bytes.

Tokens:
After the 8 byte header, the file is just a bunch of tokens/records concatinated together one right after another. Unlike many other spreadsheet formats, Mesa 2 uses very few tokens. At this time, there are only 26 different tokens defined, and 6 of them are not used at all in the "Mesa JDK" file format but are defined for compatibility with older files. Most other spreadsheet file formats have many tokens, sometimes up to several hundred. Mesa 2 is able to use fewer tokens by making it's tokens adaptible. The

data at the end of the token may depend on data found at the beginning of a token. For example, in many spreadsheets, a cell that contains a formula would be written in a different token than a cell that only contains a number. In Mesa 2, these both would be stored in a "CELL" record and flags at the begining of the CELL would let the file loader know that a formula needs to be loaded later in the cell record. Thus, very few of the Mesa tokens have a fixed size or format.

Token header format:
Every token first starts out with a single byte depicting the token type. Following the token type byte is a unsigned short (2 bytes) which give the length. However, if the high order bit of the unsigned short is set, then the unsigned short is immediately followed by another unsigned short. The lower 15 bits of the first unsigned short are used for the upper 15 bits of a 31 bit int used to describe the length. This setup allows the token header to be 3 bytes long for records under 32K in size and 5 bytes long for tokens larger than 32K. Token larger than 2GB are not supported. The length includes the 2/4 bytes for the length, but does not include the 1 byte for the token identifier.

Some sample code for reading the token headers:

```
int token = readUChar(file);
int len = readUShort(file);
if (len & 0x8000) {
     int len2 = readUShort(file);
     len &= 0x7FFF;
     len <<= 16;
     len += len2;
}
```

Note: Most of the sample code in this documentation will assume that you know C/C++ and that there are a bunch of "utility" functions set up to handle common operations. Most of these utiltity functions are fairly self explanatory. The above example uses two utility functions: readUChar and readUShort which would read an unsigned char (1 byte) and an unsigned short (2 bytes) from the file and return it.

An example token, the beginning of file record, containing 12 bytes of data would look like:

```
01 0E 00 03 00 00 00 4A 01 00 00
52 03 00 00
```

The 01 signifies the beginning of file record. 000E signifies that the record, including the 000E is 14 bytes long. 00000003 is the version number of the file. Mesa JDK files and Mesa CMP files are version 3, Mesa DPP are version 2, and Mesa2DPP are version 1. 0000014A (decimal 330) is the build number of Mesa that created this file. This is important. The Mesa JDK file format was designed with adaptibility in mind.

Some tokens will change themselves by adding information onto the end of themselves depeneding on the version. For example, the "Script" token in files before build 228 will not have information about the keyboard shortcut for the script. Files from builds after 228 will have that information.
00000352 is the codepage of the machine that created the file. These 4 bytes are only written out for files created after build 312.

This brings up another important aspect about reading Mesa files: never assume that this documentation is complete and describes ALL parts of a token. Later builds of Mesa may add information onto the end of tokens. Thus, it is very important to use the length part of the token header to determine where the next token is found. For example:

```
while (!done){
      int token = readUChar(file);
      int nextPos = getFilePosition(file);
      int len = readUShort(file);
      if (len & 0x8000) {
            int len2 = readUShort(file);
            len &= 0x7FFF;
            len <<= 16;
            len += len2;
      }
      nextPos += len;
      switch (token) {
            // process the token
      }
      seekToPosition(file,nextPos);
}
```

This "dynamic" nature of Mesa files allows us to add information to the file while retaining complete backwards and forwards compatibility. Older versions of Mesa 2 will simply ignore and skip over any information found after the parts that it knows about. Newer versions of Mesa will check the build number in the BOF token and not read in information that was not present in that build.

The Tokens:

As mentioned previously, there are 26 tokens defined for Mesa 2 files. They are as follows:

Token	Description	Byte Value
BOF	Beginning of file	1
EOF	End of file	2

CELL	A single cell ***	3
FORMAT	A single format	4
LAYER21	An old (2.1) layer *	5
CLIPBRD	Clipboard Information**	6
FONT	A Single font	7
SCRIPT21	An old (2.1) script *	8
FRAME	A graph or graphic	9
NUMFMT	Local number format	10
LABELS	Range labels	11
LAYER	New (2.2 Layer)	12
RECALC	Recalc behavior info	13
PRINTHD	2.1 print information *	14
POSITION	Window postition	15
SCRIPT	Script	16
DDEITEM	A DDE item	17
PASSWD	A password	18
ZOOM	Zoom information *	19
BLOB	A BLOB for AddIns	20
SIZE	Cached size information	21
TXTSQR	2.0.4 Text square *	22
TABSET	2.1 tab information *	23
UNDO	Undo/Redo levels	24
ROW	Row of CELL records	25
PRINTINFO	Printer information	26

Tokens marked with * are not found in 2.2 files but may be found in older versions of Mesa files.
Tokens marked with ** are not found in any Mesa file, but are used for clipboard operations.
Tokens marked with *** are not written by Mesa 2 into a 2.2 file. However, they may be more convienient to use when writing an import filter. Mesa 2 will recognize and load these tokens.

Required Tokens:

When creating a file, the only required tokens are the BOF and EOF tokens, however, that is not a very useful file. If the file does not contain at least 1 layer, format, or font token, Mesa will create a default one for each of them.

Note:

Due to a minor bug in Mesa 2.2, you need to do one of:

a) Write 2 font tokens, the first being "12.Helvetica", the second being "14.Helvetica Bold"

b) Write at least one layer token. If you don't write any font tokens, make sure

the rulerfont is set to 0.

The bugs that cause these restictions will be fixed in a future update.

Data Types:

Other than ints (4 bytes), shorts (2 bytes), doubles (8 bytes), floats(4 bytes), chars (1 byte), there are a bunch of complex data types that may be found in the tokens.

BlockLen

Many tokens makes use of a "blocklen" type to record the size of the following data. For lengths less than 32K, the blocklen structure is only 2 bytes. If it is larger than 32K, the blocklen is 4 bytes. When reading this structure, read a short first. If the hi order bit is set, then read the second short and combine them.

```
int len = readUShort(file);
if (len & 0x8000) {
     int len2 = readUShort(file);
     len &= 0x7FFF;
     len <<= 16;
     len += len2;
}
```

When writing a blocklen token, you may always use the 4 byte type if you would like. For example:

```
writeUShort(((len >> 16) & 0xFFFF) | 0x8000);
writeUShort(len & 0xFFFF);
```

Address

An address is a data structure that describes a single address in the spreadsheet. It contains a row (4 bytes), a column (2 bytes), and a layer (2 bytes). However, to optimize storage requirements, it may be stored in a smaller format. If the highest bit of the layer (bit 15) is set, the address is a "null" address or undefined address. This is mostly used in ranges that have an upper left but no lower right.

Name	Type	Comment
flags	char	Flags describing the address

If flags is 0, the rest of the address is:

Name	Type	Comment
layer	short	The layer
column	short	The column

row	int	The row

If the high bit of flags is set:

Name	Type	Comment
column	char	The column
row	short	The row

The layer is stored in the lower 7 bits of flags. This format is used for cells where the layer is <= to 127, the row is <= 65535, and the column is <= 255.

If the highest bit is not set and the second highest bit of flags is set:

Name	Size	Comment
column	short	The column
row	short	The row

The layer is stored in the lower 6 bits of flags. This format is used for cells where the layer is <= to 63, the row is <= 65535, and the column is <= 65535.

A sample utility function for reading an address from a file would look something like:

```
void readAddress(file, int &row, int &col, int &layer)
{
     int flag = readUChar(file);
     if (flag & 0x80) {
          layer = flag & 0x7F;
          column = readUChar(file);
          row = readUShort(file);
     } else if (flag & 0x40) {
          layer = flag & 0x3F;
          column = readUShort(file);
          row = readUShort(file);
     } else {
          layer = readUShort(file);
          column = readUShort(file);
          row = readInt(file);
     }
}
```

When writing an address to a file, it is usually easiest to only use the full 9 byte format. Just write 0 for the first byte, then the layer as a short, then the column as a short, and then the row as an int. This foes make ther resulting file a bit larger, but it reduces the complexity of determining how to write the address.

Range

A range is basically a collection of Addresses along with some extra flags.

Name	Type	Comment

count/flags	int	The count of ul/lr pairs (lower 16bits) and extra flags (upper 16 bits) 0x10000 - autogrow 0x20000 - automove

Each pair of ul/lr address is defined as:

Name	Type	Comment
ul	Address	The upper left cell of the range
lr	Address	The lower right cell of the range

Selection

A selection represents all the items that are currently selected.

Name	Type	Comment
len	blocklen	The size of the data in this token
layer	int	The current layer
address	Address	The current base cell
upperleft	Address	The upperleft cell of the view
range	Range	The currently selected range
ngraphs	int	The number of selected graphs
graphnames	String...	The graph names (must have ngraphs number of strings)
script	String	The name of the currently selected script
numsel	int	The number of embedded selections (for other layers)
embedsel	Selection	The embedded selections
splitul	Address	The upper left cell of the split (locked) area
splitlr	Address	The lower right edge of the split (locked) area
smin	int	The index into the script of the start of the selection
smax	int	The index into the script of the end of the selection
stop	int	The index into the script of the first character

Rect

This structure describes a rectangle used in various places in the UI.

Name	Type	Comment
x	int	The X coordinate
y	int	The Y coordinate
wid	int	The width
hi	int	The height

Color

A Color is an RGB representation of a color.

Name	Type	Comment
color	int	RGB color, high order byte is 0

String

A text string

Name	Type	Comment
len	blocklen	the length (including NULL) of the string
string	varies	the string itself

The len is encoded exactly like the length in the token headers using 2 bytes for strings less than 32K in size and 4 bytes for longs strings. The len might be 0xFFFFFFFF in which case, the string is NULL. This is different than an empty string whcih would have a len of 1.

Font

The font structure completely describes a font used on the display.

Name	Type	Comment
fontsize	int	The size of the font in points
len	short	The length of the font name
name	varies	The font name

Font attributes like Bold and Italic are encoded into the name. For example, a bold Helvetica font would be recorded as "Helvetica Bold".

Ruler

The ruler structure defines the attributes for the rows or columns.

Name	Type	Comment
len	blocklen	the length of this data structure
size	short	the default size, in points

After the default size, the Ruler contains the information for each row or column in a compressed form.

Name	Type	Comment
index	int	The row or column number is in lower 31 bits.
size	short	the size in points in lower 12 bits
format	short	the default format for the row/column (must be 0)

If index is 0xFFFFFFFF, then there are no more row/column record information and size and format do not exist for that record. The size is defined as follows:

Name	Mask	Comment
points	0x0FFF	The size in points
hidden	0x4000	The row/column is hidden
pgbreak	0x8000	The row/column is a page break.

If the high order bit of index is set, then the format bytes are immediately followed by:

Name	Type	Comment
count	short	The count of consecutive rows/columns that are identically formatted to this one.

This is a space savings optimization. If you are creating a Mesa 2 file, you do not need to compress identical rows/columns down into one token.

Image

An image structure is used to define all images in Mesa 2. Currently, there are only two image types defined, Bitmap and OS/2 Metafile. Any token in the file that allows an Image can take either a Bitmap or an OS/2 Metafile and properly display it.

Name	Type	Comment
type	int	The image type. Currently only 0 (Bitmap) and 1 (OS/2 Metafile) are used.

If the type is 1 (OS/2 Metafile), the rest of the token is:

Name	Type	Comment
len	int	The length of the metafile data
data	varies	The Metafile data

The data is the data returned from GpiQueryMetafileBits and can be used with GpiSetMetaFileBits to set the bits for a new Metafile when loaded.

If the type is 0 (Bitmap), the rest of the token is:

Name	Type	Comment
headerlen	int	the length of the bitmap header
bitslen	int	the length of the bits
header	varies	the BITMAPINFOHEADER2 structure
bits	varies	the bits for the bitmap

The header and bits can directly be used in GpiCreateBitmap and GpiSetBitmapBits to create a bitmap in memory.

PrintInfo

The PrintInfo data structure is the structure that stores all the information about the print settings for the layer.

Name	Type	Comment
len	blocklen	the length of this data structure
headertl	String	the top left header
headertc	String	the top center header
headertr	String	the top right header
headerbl	String	the bottom left header
headerbc	String	the bottom center header
headerbr	String	the bottom right header

headerl	String	the left header
headerr	String	the right header
flags	short	print flags
scale	short	the print scale
marginleft	short	the left margin (points)
margintop	short	the top margin (points)
marginright	short	the right margin (points)
marginbtm	short	the bottom margin (points)

The flags item describes some of the options that can be set:

Name	Mask	Comment
grid	0x001	Print grid if set
rchead	0x002	Print row/column headers if set
order	0x004	Down then over if set
fitv	0x008	Fit to page vertically if set (ignores scale)
fith	0x010	Fit to page horizontally if set (ignores scale)
landscape	0x020	Print in landscape mode if set
centerh	0x040	Center horizonatally on page
centerv	0x080	Center vertically on page
notes	0x100	Print notes for this layer

Record descriptions:

As described before, every token/record begins with the token ID byte and the 2/4 bytes used to record the length of the token. The data in the tokens will be describes in the next sections. Each part of the data will be given a name, a size (in bytes) and a comment.

BOF

The beginning of file record marks the start of the data as well as provides information about the source of the file. This should be the very first token in the file found immediately after the 8 byte header.

Name	Type	Comment
version	int	File version (00000003 for 2.2 files)
build	int	Mesa 2 build number (document describes build 331)
codepage	int	Source codepage

The codepage part is only found in files created after build 312.

EOF

Marks the end of the file. Should be the last token in the file. This token contains no data.

CELL

This token contains data that describes a single cell. It is not written out to a Mesa 2.2 file. Instead, Mesa 2 uses the **ROW** token to describe an entire row of cells at once. The **ROW** token is much more space efficient as it saves up to 7 bytes per cell. In files with lots of cells, this can add up quickly. However, Mesa 2 will recognize the CELL token in files and will load it. When writing a filter from one spreadsheet file format to Mesa 2 format, it is usually more convenient to use the CELL token than the ROW token.

Name	Type	Comment
address	Address	The cells address - see Address data type
flags	short	The flags describing the data in the cell
format	short	The index into the format palette

The rest of the CELL record varies depending on the value of flags.

Name	Mask	Comment
type	0x00F	The value type for the cell
comment	0x010	The cell has a comment/attachment
hasrtf	0x020	The cell has a rtf formatting information
formula	0x040	The cell has a formula
numtype	0x180	The number type if type specifies a number

If the flags specify that the cell contains a formula, then the format index is immediately followed by the RPN stream of bytes representing the formula. The format for the RPN stream will be described later in this document.

Name	Type	Comment
size	short	The RPN length, including these 2 bytes
rpn	varies	The RPN data. Length of (size - 2)

The size is an unsigned short. That allows very complex formulas with the RPN size up to 64K in size.

After the RPN stream (or after the format index if the cell does not have an RPN stream) is where the current value of the cell is stored. The format for this depends on the "type" field in the flags.

0 - Error Value Type

Name	Type	Comment
error	short	The error number

1 - String Value Type

Name	Type	Comment

string	String	The string

If the Flags for the cell indicate that there is RTF information available for the string, that information immediately follows the string. This information is documented later.

Name	Type	Comment
len	blocklen	The length of the RTF information
rtf	varies	The rtf information

2 - Number Value Type

The number type uses the "numtype" bits in the flags to determine the type of number that was saved. Mesa 2 tries to store a number in as few bytes as possible.

0 - the number is stored as a signed short (2 bytes)
1 - the number is stored as a signed int (4 bytes)
2 - the number is stored as an IEEE float (4 bytes)
3 - the number is stored as an IEEE double (8 bytes)

3 - Array Value Type

This will be documented at a later time.

After the current value, if the flags denote that this cell contains a comment or attachment, the attachement is stored.

Name	Type	Comment
type	int	The type of the attachment
size	int	The length of the attachment
atttachment	varies	The attachment data

At this time, only attachment type 1 is defined and signifies a string. Other types may be reserved for future use.

One note about the order of the CELL records: it's not important. Mesa 2 will write the CELLS in the file in reverse order starting with the last cell in the file and going left/up. This is to optimze file load times as none of the arrays will need to be expanded. When creating a Mesa 2 file, write the CELL records in whatever order is easiest. The only problem may be that it takes a tiny bit longer to load.

FORMAT

The FORMAT token describes all of the visual formatting information for a cell.

Name	Type	Comment
textcolor	Color	The Text color for the cell

bkgcolor	Color	The background color for the cell
patcolor	Color	The foreground color for the pattern
ltborder	Color	The color of the left border
rtborder	Color	The color of the right border
topborder	Color	The color of the top border
bmborder	Color	The color of the bottom border
font	short	The index into the font palette
numformat	short	The number format bit flags
info	int	The extra formatting information
pattern	char	The pattern for the cell
ltbordert	char	The left border type
rtbordert	char	The right border type
topbordert	char	The top border type
bmbordert	char	The bottom border type
underline	char	The underline style

If the numformat specifies an extended or custom number format, the underline is immediately followed by:

Name	Type	Comment
extformat	String	The number format expressed as text

If the alignment is "SpanColumns", there is 1 more byte in the token:

Name	Type	Comment
spncols	1	The number of columns to span

The info field is a bunch of bits defined as:

Name	Mask	Comment
halignment	0x00007	The horizontal alignment 0 - generalAlignment 1 - leftAlignment 2 - rightAlignment 3 - centerAlignment 4 - spanColumnsAlignment
valignment	0x00018	The vertical alignment 0 - vertBottomAlignment 1 - vertTopAlignment 2 - vertCenterAlignment
hiddenzero	0x00020	Hide zero values
wrap	0x00040	Wrap the strings
protected	0x00080	The cell is protected
inputtype	0x00F00	The input type 0 - allInputType

		1 - numbersInputType 2 - stringsInputType 3 - datesInputType 4 - formulasInputType
clearbkg	0x01000	The bkg is clear (don't use bkgcolor)
rednegative	0x02000	Display negative values in red
textdirection	0x1C000	The text direction
strikethrough	0x20000	Strikethrough size

The numformat field is a bunch of bits that describe the number formatting.

Name	Mask	Comment
extformat	0x8000	The number format is an extended format. The rest of the numformat field is ignored
numtype	0x0E00	The formatting type 0 - unformatted 1 - decimal 2 - currency 3 - date/time
subtype	0x01C0	Unformatted: 0 - general format 1 - text format 2 - hidden format Decimal: 0 - fixed 1 - scientific 2 - percent Currency: 0 - currencysubtype 1 - comma Date/time: 0 - date format 1 - time format

For currency types (currency and comma):

Name	Mask	Comment
selector	0x0030	The currency type selector

For decimal and currency types:

Name	Mask	Comment
decimals	0x000F	The number of decimal places

For time formats:

Name	Mask	Comment
timeformat	0x003F	The time format 0 - HMS12 1 - HM12 2 - HMS24 3 - HM24 4 - System/country defined time format

For date formats:

Name	Mask	Comment
dateformat	0x003F	The date format 0 - MM/DD/YY 1 - MM/DD 2 - DD-MMM 3 - DD-MMM-YY 4 - MMM-YY 5 - YY/MM/DD 6 - DDMonYY 7 - MonDDYY 8 - DDMon 9 - MMDDYY 10 - MMDD 11 - DDMM 12 - DDMMYY 13 - MMM-DD-YY 14 - MMM-DD 15 - MMM DD, YYYY 16 - DDMMYYYY 17 - MMDDYYYY 18 - MMYYYY 19 - YYMMDD 20 - DD_MON_YYYY 21 - MON_DD_YYYY 22 - MON_YYYY 23 - DD_MM_YYYY 24 - MM_DD_YYYY 25 - DD_MM 26 - MM_DD 27 - MM_YYYY 28 - DD_MM_YY 29 - MM_DD_YY 30 - DD/MM/YYYY 31 - MM/DD/YYYY

32 - DD/MM
33 - MM/YYYY
34 - DD/MM/YY
35 - DD.MM.YYYY
36 - MM.DD.YYYY
37 - DD.MM
38 - MM.DD
39 - MM.YYYY
40 - DD.MM.YY
41 - MM.DD.YY
42 - DDxMMxYYYY
43 - MMxDDxYYYY
44 - DDxMM
45 - MMxDD
46 - MMxYYYY
47 - DDxMMxYY
48 - MMxDDxYY
49 - System defined date format
50 - YYxMMxDD
51 - YY.MM.DD
52 - YY_MM_DD

LAYER21

This token does not appear in 2.2 files and is thus not documented.

CLIPBRD

This token does not appear in 2.2 files and is thus not documented.

FONT

This record describes a single font. Font numbering is 0 based. When the file is loaded, the first font found is given number 0, the second font is given number 1, etc...

Name	Type	Comment
fnt	Font	The font structure

SCRIPT21

This token does not appear in 2.2 files and is thus not documented.

FRAME

This token is not yet documented

NUMFMT

This token specifies the decimal separator, thousands separator, leading string,

and tailing string for the CURRENCY number formats (ser FORMAT token).

Name	Type	Comment
decsep0	int	the decimal separator
thoussep0	int	the thousands separator
leadstring0	String	the leading string
tailstring0	String	the tailing string
decsep1	int	the decimal separator
thoussep1	int	the thousands separator
leadstring1	String	the leading string
tailstring1	String	the tailing string
decsep2	int	the decimal separator
thoussep2	int	the thousands separator
leadstring2	String	the leading string
tailstring2	String	the tailing string
decsep3	int	the decimal separator
thoussep3	int	the thousands separator
leadstring3	String	the leading string
tailstring3	String	the tailing string

LABELS

This token defines all of the range labels.

Name	Type	Comment
numlabels	int	Number of defined labels

For each label that is defined:

Name	Type	Comment
name	String	Label name
rng	Range	the defined range

LAYER

This record describes all the information about a particular layer. Every layer must have one of these in the file. If there is not a LAYER token in the file, Mesa 2 will assume there is only one layer and will create a default one.

Name	Type	Comment
reserved	char	Reserved - must be 0
bkgColor	Color	The background color for the layer
rlrColor	Color	The color of the row/column titles
gridColor	Color	The colr of the gridlines
font	short	Index into the font palette for the font used in the row/column headers
showgrid	char	1 - grid is on, 0 - grid is off
name	String	the layer name
protection	char	1 - the layer is protected, 0 - the layer is not protected

nrows	int	the number of rows displayed in the layer
ncols	short	the number of cols displayed in the layer
locklevel	char	reserved - must be 0
comment	String	The comment for the layer

The information beyond the comment varies.

Name	Type	Comment
hasPrintInfo	char	The layer has printer information set.
printinfo	varies	The printer information

If hasPrintInfo is 1, then printinfo exists and should be read in before going to the next byte in the token. See the PrintInfo data structure. If hasPrintInfo is 0, printinfo does not exist and the next byte in the token is:

Name	Type	Comment
hasBkgImg	char	The layer has an image
image	Image	The image

If hasBkgImg is 1, then image exists and must be read in before going on to the next byte in the token. See the Image data structure.

The rest of the Layer token looks like:

Name	Type	Comment
colRuler	Ruler	The information about the columns
rowRulter	Ruler	The information about the rows
numTitles	int	Number of column titles defined
titles	String...	The titles

RECALC

This record describes how the recalc engine is to recalc this workbook.

Name	Type	Comment
order	int	Calc order (0 - row, 1 - column, 2 - natural, 3 - natural plus circular refs)
iterations	int	Number of iterations
autocalc	int	1 if auto recalc is turned on, 0 if manual recalc
reserved	int	reserved for future use, should be 0

PRINTHD

This token does not appear in 2.2 files and is thus not documented.

POSITION

This record describes the window positions that are used when the file is restored.

Name	Type	Comment
num	char	The number of windows into the file that are open
apprect	Rect	The rectangle of the application
currect	Rect	reserved

For each of the num windows, there is:

Name	Type	Comment
winRect	Rect	The window rectangle relative to apprect
tabfont	Font	The font used for the tabs
disptab	char	1 if tabs are visible, 0 if not
zoom	short	The current zoom scaling
minmax	char	1 if minimized, 2 if maximized, 0 if neither
sel	Selection	The selection for the window

SCRIPT

This token contains a REXX script.

Name	Type	Comment
name	String	The name of the scipt
script	String	The actual text of the script
info	int	Information bit flags
comment	String	The script comment
hasIcon	char	1 if the icon is present. 0 if it isn't.
icon	Image	The icon for the toolbar. (Bitmap image only)
key	int	The keyboard shortcut
modifiers	short	The keyboard modifier flags

Note: the key and modifiers fields are only present in files created after build 328.

The info bits are defined as:

Name	Mask	Comment
onopen	0x01	Execute this script when file is opened
onclose	0x02	Execute this script when file is closed

The key and modifiers fields are undocumented.

DDEITEM

This token records the information required to reconnect to a DDE data server, retreive the information, and place it in the proper location in the spreadsheet.

Name	Type	Comment
app	String	The name of the DDE server application
file	String	The name of the file
item	String	The item name
name	String	The name of this DDE item

selection	Selection	The selection into which to paste the DDE data
type	short	The link type
		0 - Hot link
		1 - Warm link
		2 - Cold link
ack	short	The force acknowledge flag
format	int	The source data format
		2 - Text (Tabular)
		5 - Private (only if app is Mesa2)

PASSWD

For security reasons, this token is not documented.

ZOOM

This token does not appear in 2.2 files and is thus not documented.

BLOB

This token is only used by AddIns to store information that they need. BLOB stands for binary length of bytes.

Name	Type	Comment
name	String	The name
size	int	The length of the BLOB data
blob	varies	The BLOB data

SIZE

This token is usually stored immediately after the BOF record. It is used to record the size of some of the internal arrays. Having this record allows Mesa 2 to pre-allocate some of the arrays, thus speeding up file loading.

Name	Type	Comment
numcells	int	Not used in 2.2
numformats	int	The number of defined formats
numfonts	int	The number of defined fonts
numlayers	int	The number of layers
numscripts	int	The number of scripts
numframes	int	The number of frames/graphics
numblobs	int	The number of blobs.

Note: These do not actually create the specified number of objects. They only make sure the room for them exists. You still need to define the appropriate tokens.

TXTSQR

This token does not appear in 2.2 files and is thus not documented.

TABSET

This token does not appear in 2.2 files and is thus not documented.

UNDO

Stores the number of UNDO/REDO levels set for this workbook.

Name	Type	Comment
undo	int	Number of UNDO levels.

ROW

An optimized group of cell records. Mesa 2 version 2.2 uses this instead of CELL records to save space in the file.

Name	Type	Comment
layer	short	The layer for all cells in this record
row	int	The row for all cells in this record

After the row is a bunch of cells as follows

Name	Type	Comment
column	short	The column number. If equal to 0xFFFF, there is no more data following this.
len	blocklen	The length of the data for this cell.
celldata	varies	The cell data

The celldata is exactly like the information stored in the CELL record except that the Address structure is not written out as it is not needed.

One note: the order of the cells in the record is not important. For file loading speed optimizations, Mesa 2 writes the cells out in reverse order. In other words, the last column in the row is written first, then the second to the last, etc.... Putting them in sequence is allowed and will work. Also, Mesa 2 writes the last row on the layer first and then the next to last, etc... This order is also not important, but the reverse order allows the file to load a tiny bit faster.

PRINTINFO

This token stores the platform specific printer information used by the printer drivers on that platform.

Name	Type	Comment
platform	char	The platform - currently, only 0 (OS/2) is defined
name	String	The name of the printer that was last used to print the file
datalen	int	The length of the printer specific data

data	varies	The data

RPN Byte Stream

Mesa 2 stores formulas in a tokenized RPN (Reverse Polish Notation) form. Each token in the RPN stream is 2 bytes, but many tokens are immediately followed by additional bytes of information.For example, the "push addresss" token would immediately be followed by the bytes of data needed to construct an Address structure. Mesa 2.2 currently defines 540 tokens.

The majority of the tokens that are in the RPN stream refer to formula functions. They are defined in the following table. For all functions, if the minimum number of parameters does not equal the maximum number of parameters, the token is immediately followed by another 2 bytes (a short) specifying the number of parameters that the user typed for that formula. For example, the AVE part of the formula AVE(num1,num2) would look like

19 00 02 00

A note about obsolete formulas: some formulas have changed since 2.0 shipped. For example, the HLOOKUP function has an optional parameter added to it. Rather than make a new version of the same token, a new token was added. Thus there are multiple tokens for some formula functions. In these cases, the safest thing to use is the LAST one in the table. The first entries are considered obsolete. While Mesa can calculate with them, it is advised that you not use them.

Token	Formula Name	Min Params	Max Params
3	SIN(	1	1
4	LENGTH(	1	1
10	NOT(	1	1
12	IF(	2	3
17	FEED(	2	2
20	SIGNAL(	3	3
24	ABS(	1	1
25	AVE(	1	64
26	COS(	1	1
27	COUNT(	1	64
28	EXP(	1	1
29	FRAC(	1	1
30	INT(	1	1
31	MAX(	1	64
32	MIN(	1	64

33	PROD(	1	64
34	RADTODEG(	1	1
35	ROOT(	2	2
36	ROUND(	2	2
37	SGN(	1	1
38	SQRT(	1	1
39	SUM(	1	64
40	CALCRATE(	3	3
41	@FV(	3	3
42	IRR(	1	2
43	NPV(	2	64
58	(	1	1
59	MOD(	2	2
60	@PMT(	3	3
61	@PV(	3	3
62	@RATE(	3	3
63	VALUE(	1	1
64	DEGTORAD(	1	1
67	@CTERM(	3	3
68	@TERM(	3	3
69	TAN(	1	1
70	@TAN(	1	1
71	VARP(	1	64
72	STDEVP(	1	64
73	STDDEVP(	1	64
74	NEXT(	1	1
75	@NEXT(	1	1
76	@@(	1	3
77	ADDRESS(	1	3
78	ISEMPTY(	1	1
79	ISFORMULA(	1	1
80	ISSTRING(	1	1
81	ISNUMBER(	1	1
82	ISARRAY(	1	1
83	ISNA(	1	1
84	ISERROR(	1	1
85	SAME(	1	1
86	@SAME(	1	1
87	AVERAGE(	1	64
88	AVG(	1	64
89	SUMSQ(	1	64
90	NUMBER(	1	1
91	@SIN(	1	1

92	@SUM(	1	64
93	@PROD(	1	64
94	@AVE(	1	64
95	@AVG(	1	64
96	@AVERAGE(	1	64
97	@COUNT(	1	64
98	@IF(	2	3
99	IFELSE(	3	3
100	@NOT(	1	1
101	@MAX(	1	64
102	@MIN(	1	64
103	@VAR(	1	64
104	STD(	1	64
105	@STD(	1	64
106	@STDEV(	1	64
107	@STDDEV(	1	64
108	@ABS(	1	1
109	@SUMSQ(	1	64
110	SIGN(	1	1
111	@SGN(	1	1
112	@SIGN(	1	1
113	@ROUND(	2	2
114	@INT(	1	1
115	@SQRT(	1	1
116	@MOD(	2	2
117	DIV(	2	2
118	@DIV(	2	2
119	LOG(	1	1
120	LOG10(	1	1
121	@LOG(	1	1
122	@EXP(	1	1
123	LN(	1	1
124	@LN(	1	1
125	@COS(	1	1
126	ASIN(	1	1
127	@ASIN(	1	1
128	ACOS(	1	1
129	@ACOS(	1	1
130	ATAN(	1	1
131	@ATAN(	1	1
132	ATAN2(	2	2
133	@ATAN2(	2	2
134	PV(	3	5

135	@NPV(	2	64
136	FV(	3	5
137	PMT(	3	5
138	@VALUE(	1	1
139	@NUMBER(	1	1
140	EXACT(	2	2
141	@EXACT(	2	2
142	REPT(	2	2
143	REPEAT(	2	2
144	@REPT(	2	2
145	@REPEAT(	2	2
146	LEN(	1	1
147	@LENGTH(	1	1
148	@LEN(	1	1
149	@MID(	3	3
150	@MID(	3	3
151	ISERR(	1	1
152	@ISERROR(	1	1
153	@ISERR(	1	1
154	@ISNA(	1	1
155	ISREF(	1	1
156	@ISREF(	1	1
157	INDEX(	1	4
158	@INDEX(	1	4
159	@FRAC(	1	1
160	CHAR(	1	1
161	@CHAR(	1	1
162	CLEAN(	1	1
163	@CLEAN(	1	1
164	CODE(	1	1
165	@CODE(	1	1
166	LEFT(	2	2
167	@LEFT(	2	2
168	RIGHT(	2	2
169	@RIGHT(	2	2
170	UPPER(	1	1
171	@UPPER(	1	1
172	LOWER(	1	1
173	@LOWER(	1	1
174	PROPER(	1	1
175	@PROPER(	1	1
176	FIND(	2	3
177	@FIND(	2	3

178	@ISSTRING(	1	1
179	REGRESS(	2	2
180	@REGRESS(	2	2
181	@STRING(	1	2
182	FIXED(	1	2
183	@ISNUMBER(	1	1
184	ISTEXT(	1	1
185	@N(	1	1
186	N(	1	1
187	@S(	1	1
188	S(	1	1
189	DOLLAR(	1	2
190	@CHOOSE(	2	64
191	@CHOOSE(	2	64
192	@HLOOKUP(	3	3
193	@HLOOKUP(	3	3
194	@VLOOKUP(	3	3
195	@VLOOKUP(	3	3
196	TRIM(	1	1
197	@TRIM(	1	1
198	SYD(	4	4
199	@SYD(	4	4
200	SLN(	3	3
201	@SLN(	3	3
202	DDB(	4	5
203	@DDB(	4	5
204	@ROOT(	2	2
205	GETINPUT(	1	1
206	@GETINPUT(	1	1
207	@REPLACE(	4	4
208	@REPLACE(	4	4
209	@IRR(	2	64
210	@SIGNAL(	3	3
211	@ISEMPTY(	1	1
212	@ISFORMULA(	1	1
213	@RADTODEG(	1	1
214	@DEGTORAD(	1	1
215	@ADDRESS(	1	3
216	@CALCRATE(	3	3
217	CTERM(	3	3
218	ISTRING(	1	1
219	@ISTRING(	1	1
220	DATE(	3	6

221	@DATE(	3	6
222	TIME(	3	3
223	@TIME(	3	3
224	YEAR(	1	1
225	@YEAR(	1	1
226	MONTH(	1	1
227	@MONTH(	1	1
228	DAY(	1	1
229	@DAY(	1	1
230	HOUR(	1	1
231	@HOUR(	1	1
232	MINUTE(	1	1
233	@MINUTE(	1	1
234	SECOND(	1	1
235	@SECOND(	1	1
242	INVERT(	1	1
243	@INVERT(	1	1
244	TRANSPOSE(	1	1
245	@TRANSPOSE(	1	1
246	MULT(	2	2
247	@MULT(	2	2
279	ELEMENT(	3	3
280	@ELEMENT(	3	3
281	LASTROW(	1	1
282	@LASTROW(	1	1
283	LASTCOL(	1	1
284	@LASTCOL(	1	1
285	@DOLLAR(	1	2
287	@FEED(	2	2
288	STRING(	1	2
289	@FIXED(	1	2
290	@IFELSE(	3	3
291	@ISARRAY(	1	1
292	@ISTEXT(	1	1
295	TERM(	3	3
296	RATE(	3	6
297	NPER(	3	5
302	SINH(	1	1
303	@SINH(	1	1
306	STDDEV(	1	64
307	@STDS(	1	64
308	STDEV(	1	64
309	SUMPRODUCT(	1	64

310	@SUMPRODUCT(	1	64
311	VAR(	1	64
312	@VARS(	1	64
313	RUNSCRIPT(	2	2
314	@RUNSCRIPT(	2	2
315	DATEVALUE(	1	1
316	@DATEVALUE(	1	1
317	TIMEVALUE(	1	1
318	@TIMEVALUE(	1	1
319	ISBLANK(	1	1
320	MINVERSE(	1	1
321	MMULT(	2	2
322	DEGREES(	1	1
323	PRODUCT(	1	64
324	RADIANS(	1	1
325	TEXT(	1	2
326	@ISRANGE(	1	1
327	@ISNAME(	1	1
328	ISRANGE(	1	1
329	ISNAME(	1	1
330	@CELL(	1	2
331	CELL(	1	2
332	@CELLPOINTER(	1	1
333	CELLPOINTER(	1	1
334	@ROWS(	1	1
335	ROWS(	1	1
336	@COLS(	1	1
337	COLS(	1	1
338	@SHEETS(	1	1
339	SHEETS(	1	1
340	@COORD(	3	4
341	COORD(	2	4
342	@ISLABEL(	1	1
343	ISLABEL(	1	1
344	LAYERS(	1	1
345	@LAYERS(	1	1
346	@TEXT(	1	2
347	@DEGREES(	1	1
348	@RADIANS(	1	1
349	@MMULT(	2	2
350	@MINVERSE(	1	1
351	@ISBLANK(	1	1
352	DSUM(	3	3

353	@DSUM(	3	3
354	DAVERAGE(	3	3
355	@DAVERAGE(	3	3
356	DAVE(	3	3
357	@DAVE(	3	3
358	DAVG(	3	3
359	@DAVG(	3	3
360	DCOUNT(	3	3
361	@DCOUNT(	3	3
362	DMAX(	3	3
363	@DMAX(	3	3
364	DMIN(	3	3
365	@DMIN(	3	3
366	DSTDEV(	3	3
367	@DSTDEVS(	3	3
368	DSTDEVP(	3	3
369	@DSTD(	3	3
370	DPROD(	3	3
371	@DPROD(	3	3
372	DSUMSQ(	3	3
373	@DSUMSQ(	3	3
374	DVAR(	3	3
375	@DVARS(	3	3
376	DVARP(	3	3
377	@DVAR(	3	3
378	DPRODUCT(	3	3
379	@DPRODUCT(	3	3
380	@DSTDS(	3	3
381	DSTDDEV(	3	3
382	DSTDDEVP(	3	3
416	AND(	1	64
417	@AND(	1	64
418	OR(	1	64
419	@OR(	1	64
420	WEEKDAY(	1	2
421	@WEEKDAY(	1	2
424	VLOOKUP(	3	3
425	HLOOKUP(	3	3
426	CHOOSE(	2	64
429	TRUNC(	1	2
430	@TRUNC(	1	2
431	FLOOR(	1	2
432	@FLOOR(	1	2

433	CEILING(	1	2
434	@CEILING(	1	2
435	MROUND(	1	2
436	@MROUND(	1	2
437	ROUNDUP(	1	2
438	@ROUNDUP(	1	2
439	ROUNDDOWN(	1	2
440	@ROUNDDOWN(	1	2
441	EVEN(	1	1
442	@EVEN(	1	1
443	ODD(	1	1
444	@ODD(	1	1
445	COSH(	1	1
446	@COSH(	1	1
447	TANH(	1	1
448	@TANH(	1	1
449	ACOSH(	1	1
450	@ACOSH(	1	1
451	ATANH(	1	1
452	@ATANH(	1	1
453	ASINH(	1	1
454	@ASINH(	1	1
455	VLOOKUP(	3	4
456	@VLOOKUP(	3	4
457	HLOOKUP(	3	4
458	@HLOOKUP(	3	4
459	RUNSCRIPT(	2	64
460	@RUNSCRIPT(	2	64
461	DDELINK(	4	4
462	@DDELINK(	4	4
463	LINEST(	2	4
464	@LINEST(	2	4
465	CORREL(	2	2
466	@CORREL(	2	2
467	INTERCEPT(	2	2
468	@INTERCEPT(	2	2
469	RSQ(	2	2
470	@RSQ(	2	2
471	PEARSON(	2	2
472	@PEARSON(	2	2
473	SLOPE(	2	2
474	@SLOPE(	2	2
475	STEYX(	2	2

476	@STEYX(	2	2
477	TREND(	1	4
478	@TREND(	1	4
479	MEDIAN(	1	64
480	@MEDIAN(	1	64
481	DEVSQ(	1	64
482	@DEVSQ(	1	64
483	AVEDEV(	1	64
484	@AVEDEV(	1	64
485	REGRESS(	2	4
486	@REGRESS(	2	4
487	SMARTRANGE(	0	2
488	@SMARTRANGE(	0	2
489	SMARTRNG(	0	2
490	@SMARTRNG(	0	2
491	FREQUENCY(	2	2
492	COUNTIF(	2	2
493	SUMIF(	2	3
495	ROW(	0	1
496	COLUMN(	0	1
497	INDIRECT(	1	2
498	PERCENTILE(	2	2
500	COUNTBLANK(	1	1
501	SUBTOTAL(	2	2
502	COUNTA(	1	64
503	GAMMA(	1	1
504	BESSELY(	2	2
506	BESSELJ(	2	2
508	ERF(	1	2
509	ERFC(	1	1
510	GAMMALN(	1	1
511	FINDCELL(	2	7
512	@FINDCELL(	2	7
513	MID(	3	3
514	LEFT(	1	2
515	@LEFT(	1	2
516	RIGHT(	1	2
517	@RIGHT(	1	2
518	REPLACE(	4	4
519	@FREQUENCY(	2	2
520	@COUNTIF(	2	2
521	@SUMIF(	2	3
523	@ROW(	0	1

524	@COLUMN(	0	1
525	@INDIRECT(	1	2
526	@PERCENTILE(	2	2
528	@COUNTBLANK(	1	1
529	@SUBTOTAL(	1	64
530	@COUNTA(	1	64
531	@GAMMA(	1	1
532	@BESSELY(	2	2
534	@BESSELJ(	2	2
536	@ERF(	1	2
537	@ERFC(	1	1
538	@GAMMALN(	1	1
539	GRANDTOTAL(	1	64
540	@GRANDTOTAL(	1	64

Another common type of token are the binary operators. These are the operators, such as + and -, that display with one parameter on the left and another on the right. The list of operators include:

Token	Operator
1	+
2	-
5	&
19	*
21	>
22	<
23	/
44	=
45	!=
46	<=
47	>=
48	^
49	&&
50	#AND#
51	<AND>
52	||
53	#OR#
54	<OR>
55	^^
56	#XOR#
57	<XOR>
286	<>

299	\\=
300	\\>
301	\\<
383	=>
384	=<

There are only 3 binary operators. Binary operators only take 1 parameter and act upon that.

Token	Operator
9	-
11	+
65	#NOT#

Another common token type are constants. Constants are functions that do not take any parameters. They just return a set value or a value that does not depend on any extra parameters.

Token	Operator
6	RAND
7	RAND()
8	UNAME
236	NOW
237	@NOW
238	NOW()
239	@NOW()
240	TODAY
241	@TODAY
249	RANDOM
250	@RANDOM
251	RANDOM()
252	@RANDOM()
253	PI
254	TRUE
255	FALSE
256	@PI
257	@TRUE
258	@FALSE
259	PI()
260	TRUE()
261	FALSE()
262	@PI()
263	@TRUE()

264	@FALSE()
265	FILENAME
266	PATHNAME
267	THISROW
268	THISCOL
269	THISLAYER
270	THISADDRESS
271	CURRENTVALUE
272	@FILENAME
273	@PATHNAME
274	@THISROW
275	@THISCOL
276	@THISLAYER
277	@CURRENTVALUE
278	@THISADDRESS
293	@RAND
294	@RAND()
298	@UNAME
422	TODAY()
423	@TODAY()
427	NA()
428	@NA()

Almost all of the remaining tokens involve pushing standard value types onto the stack. These include things like numbers, addresses, ranges, strings, etc....

Token	**Description**
0	Push a IEEE 8 byte double precision number onto the stack. The token is immediately followed by the 8 bytes used to create the number.
13	Push a string onto the stack. The token is immediately followed by the null terminated string. If the string len (including the null) is not evenly divisible by 2, then an extra null is added.
16	Pushes an array onto the stack. (not yet documented)
248	Pushes an error value onto the stack. The token is immediately followed by a short depicting the error number.
305	Pushes a named range (labeled range) onto the stack. The token is followed by an unsigned short which is reserved for internal use and then the string. (see token 13)

385 Pushes a 4 byte signed integer onto the stack. The token is immediately followed by the 4 bytes.

386 Pushes a 2 byte signed short onto the stack. The token is immediately followed by the 2 bytes.

390 Pushes a IEEE 4 byte single precision float onto the stack. The token is immediately followed by the 4 bytes.

Mesa has several different formats for saving Addresses in the RPN stream. Originally, Mesa only had 1 format which was 8 bytes of additional data. In 2.1, Several smaller forms were added to save space which results in smaller files as well as less memory overhead. When creating a Mesa file, feel free to use only the full address token. The only penalty will be larger files and a larger memory requirement.

18 Full address: The token is followed by an signed short for the layer, a signed short for the column, and a signed int for the row. The upper 4 bits of the layer contains the flags for which sections of the address are absolute. The 11th bit is then copied into the upper 4 bits to extend the sign of the layer. (Remember, relative addresses can be negative)

Bit 14 - if set, the layer is absolute, not relative
Bit 13 - if set, the column is absolute, not relative
Bit 12 - if set, the row is absolute, not relative

387 Short Relative Address: This is used for addresses where all three parts are relative, the layer is between -127 and 127, the column is between -127 and 127, and the row is between -32767 and 32767. In this case, the token is followed by two signed chars and then a signed short. The first signed char is the layer and the second is the column. The signed short is the row.

388 Short Absolute Address: This is used for addresses where all three parts are absolute, the layer is less than 256, the column is less than 256, and the row is less than 65536. The token is followed by an unsigned short (the row), an unsigned char (the column), and another unsinged char (the layer).

389 Very Small Address: This is used for addresses where all three parts are relative, the layer is 0, and both the row and column are between -127 and 127. In this case, the token is followed by two signed chars. The first is the row and the second is the column.

391 Mixed Small Address: This is used for addresses where the row and column are both absolute but the layer isn't. Also, the layer between -127 and 127, the column is less than 256, and the row is less than 65536. In this case, the token is followed by an unsigned short (the row), an unsigned char (the column), and a signed char (the layer).

Like Addresses, Ranges were originally written in only one, large format. In 2.1, this was changed to create many different range tokens. The data for each range token is exactly the same. After the token, there are two addresses that define the upperleft and the lower right of the range. The only difference between the two tokens is the type of address that is written in each spot.

66 Full range - is followed by 2 Full Addresses
392 VSVS Range - is followed by two Very Small Addresses
393 VSSRel Range - is followed by a Very Small Address and then a Small Relative Address
394 VSSAbs Range - is followed by a Very Small Address and then a Small Absolute Address
395 VSRA Range - is followed by a Very Small Address and then a Mixed Small Address
396 SRelVS Range- is followed by a Small Relative Address and then a Very Small Absolute Address
397 SRelSRel Range - is followed by a Small Relative Address and then a Small Relative Address
398 SRelSAbs Range - is followed by a Small Relative Address and then a Small Absolute Address
399 SRelRA Range - is followed by a Small Relative Address and then a Mixed Small Address
400 SAbsVS Range - is followed by a Small Absolute Address and then a Very Small Address
401 SAbsSRel Range - is followed by a Small Absolute Address and then a Small Relative Address
402 SAbsSAbs Range - is followed by a Small Absolute Address and then a Small Absolute Address
403 SAbsRA Range - is followed by a Small Absolute Address and then a Mixed Small Address
404 RAVS Range - is followed by a Mixed Small Address and then a Very Small Address
405 RASRel Range - is followed by a Mixed Small Address and then a Small Relative Address
406 RASAbs Range - is followed by a Mixed Small Address and then a Small Absolute Address
407 RARA Range - is followed by a Mixed Small Address and then a Mxed

Small Address

408 NormVS Range - is followed by a Full Address and then a Very Small Address

409 NormSRel Range - is followed by a Full Address and then a Small Relative Address

410 NormSAbs Range - is followed by a Full Address and then a Small Absolute Address

411 NormRA Range - is followed by a Full Address and then a Mixed Small Address

412 VSNorm Range - is followed by a Very Small Address and then a Full Address

413 SRelNorm Range - is followed by a Small Relative Address and then a Full Address

414 SAbsNorm Range - is followed by a Small Absolute Address and then a Full Address

415 RANorm Range - is followed by a Mixed Small Address and then a Full Address

Finally, there are a few "special" tokens:

14 Old End - 2.0.x versions of Mesa used this token to denote the end of the RPN stream. 2.1 and higher just use the length of the stream to determine when the end is met.

304 AddIn Function - This is used for functions that are not recognized internally by Mesa. It is followed by two unsigned shorts. The first is the number of parameters. The second is used interally for caching. After the unsigned shorts is a NULL terminated name of the function. As with the string token, if the length of the name (including null) is not evenly divisible by 2, then an extra null is added.