Apple II
File Type Notes

Developer Technical Support

File Type: $54 (84)
Auxiliary Type: $DD3E

Full Name: Medley Desktop Publishing Document
Short Name: Medley Document

Written by: Matt Deatherage & Eric Soldan May 1989

Files of this type and auxiliary type contain documents for Medley™.

Medley is a WYSIWYG application that integrates word processing, paint, and page layout programs, with the addition of a spelling checker and thesaurus. The page layout function supports various shapes for art and text areas. Text automatically wraps around or within these areas, including irregularly shaped regions. The word processor is full-featured, as is the paint program. The dictionary has 80,000 words.

For more information on Medley, contact:

Milliken Publishing Company
1100 Research Blvd.
St. Louis, MO 63132
Attention: Medley Technical Support
(314) 991-4220

The Medley file format is copyrighted © 1988 by Milliken Publishing Company and is printed here with permission.

Definitions

The following definition is used in this document in addition to those defined for all Apple II file types:

C String A series of ASCII bytes terminated with a byte of $00. There is no count byte at the beginning, as is the case for the **String** type (also referred to as a "Pascal string").

File Structure

Medley files are basically standard, single-linked tree structures. There is a single object at the top of the tree, and other objects may branch off this parent object. Each child object is linked to the parent by a pointer to the child contained within the parent object. A non-standard thing about the Medley tree structure is that some objects may have regions or polygons associated with them. The handles to these objects are stored in the parent object when in memory, but on disk these handles are quite meaningless. Because of this difference, the regions or polygons are simply appended to the parent object itself when written to disk. The size of the region or polygon is added to the size of the parent object, giving an aggregate size for the complex object on disk.

The file is written to disk in an order based on a simple tree-walking algorithm. This algorithm starts with the highest parent object and writes it to disk. The parent object is checked for child objects. If one exists, it is written to disk, and then **it** is checked for child objects. This tree-walking continues until an object runs out of children. When that occurs, Medley backs up one tree level, writes the next child object to disk, and scans it for children. This method continues until all objects are written to disk.

For example, if a parent object named A had two child objects named B and C, where B had children E and F, and C had children G and H, the objects would be written to disk in the following order: A, B, D, E, C, F, G. Figure 1 illustrates this structure.

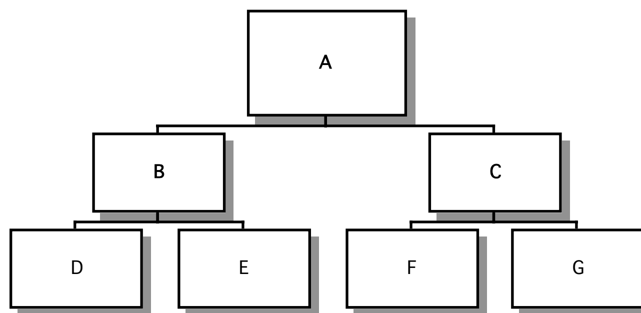

Figure 1–Example of Parent and Child Tree Structure

Some Medley objects, when in memory, have handles to other objects (such as regions or polygons) in them. Since handles are meaningless on disk, Medley stores these complex objects in an aggregate form by writing the contents of each associated handle to disk following the regular object.

The Objects and Their Formats

All objects have a common 13-byte header, which is as follows:

type	(+000)	**Byte**	The type of the object. Possible values are: 0 = Null Object (never saved to disk) 1 = Root Object (never saved to disk) 2 = File Object 3 = Page Object 4 = Paragraph Object

			5 = Area Object 6 = Art Object 10 = Document Dictionary Object. Objects 7, 8 and 9 are for posting undo events. Since these are not saved to disk, they are irrelevant for the file format.
numChildren	(+001)	**Word**	The number of children for this object. Children of the children are not included.
endData	(+003)	**Long**	The size of this object on the disk, not counting associated handles (such as regions and polygons).
reserved	(+007)	**Long**	Reserved; set to zero when creating files.
objRefNum	(+011)	**Word**	The reference number of this object, generally zero.

Each object has the header listed above followed by object-specific data. For this reason, the description of each object will start with offset +013 (the byte following the header).

The File Object

rect	(+013)	**4 Words**	Standard QuickDraw II rectangle, giving the boundary rectangle for the entire file.
pathName	(+021)	**129 Bytes**	Class zero pathname for the file on disk (used by the save command).
saved	(+150)	**Byte**	This byte is 1 if no changes have been made to the file since the last save.
windowPtr	(+151)	**Long**	Pointer to the window for this file. When creating files, set to zero.
wndwNameIndx	(+155)	**Byte**	Index into table of window names. Set to zero when creating files
windowOrigin	(+156)	**2 Words**	QuickDraw II point representing the global origin of this file's window.
windowSize.h	(+160)	**Word**	Height of window in pixels. Add to top edge of window to get bottom edge of window.
windowSize.v	(+162)	**Word**	Height of window in pixels. Add to left of window to get right edge of window.
COrigin	(+164)	**Long**	QuickDraw II point representing the scroll bar origin of this file's window. When creating files, set this to whatever origin you wish Medley to display. Make sure that the coordinate is valid.
editHndl	(+168)	**Long**	Handle to the paragraph containing the cursor. This is converted to a child number on disk, since handles on disk are meaningless.
editOffset	(+172)	**Word**	Offset to the cursor within the paragraph pointed to by editHndl.
cursor	(+174)	**4 Words**	Standard QuickDraw II rectangle, giving the rectangle used for the insert cursor. This can be set to a null rectangle when creating files, and will be calculated when the file is loaded.
showAllBorders	(+182)	**Byte**	If this is set to one, then all area borders will display, regardless of each area's border display setting.
updateRect	(+183)	**4 Words**	Standard QuickDraw II rectangle used for posting specific updates for the interruptible word processor. When creating files, set this to a null rectangle.
topMrgn	(+191)	**Fixed**	The top margin in inches.
bottomMrgn	(+195)	**Fixed**	The bottom margin in inches.
leftMrgn	(+199)	**Fixed**	The left margin in inches.

rightMrgn	(+203)	**Fixed**	The right margin in inches.
gutterMrgn	(+207)	**Fixed**	The gutter margin in inches.
pageWidth	(+211)	**Fixed**	The width of the page in inches.
pageHeight	(+215)	**Fixed**	The height of the page in inches.
selectPage	(+219)	**Word**	The active page for area editing.
numSelected	(+221)	**Word**	The number of areas currently selected by the user.
sizingDot	(+223)	**Word**	The sizing dot number of an area that was last clicked. This sizing dot will be used as the current sizing dot when the arrows are used to size an area.
effectivePage	(+225)	**Word**	The page number of the page the user was effectively editing. This is different from selectPage when the user was editing global areas at save time, for global areas are treated as on page zero.
printRecord	(+227)	**140 Bytes**	A standard IIGS Print Manager print record; the one in use for this document at save time. This field can be undefined, if the printRecordDefined field is zero.
interruptMode	(+367)	**Word**	Currently undocumented. (Set to zero.)
editScroll	(+369)	**Byte**	Currently undocumented. (Set to zero.)
firstHndl	(+370)	**Long**	Handle where wrap-around regions should actually start; i.e., where an update is needed. When creating files, set this to zero.
firstMrn	(+374)	**Word**	MiniRect number (line number of paragraph) where wrap-around regions should actually start; i.e., where an update is needed. When creating files, set this to zero.
selectMode	(+376)	**Word**	Some text is selected if this is 1. Each paragraph will indicate the range of characters selected within that paragraph. This allows the screen update routine to quickly determine which characters in a paragraph should be drawn selected.
showPgphMarks	(+378)	**Byte**	Indicates whether paragraph marks are currently being shown.
showSpaces	(+379)	**Byte**	Indicates whether spaces are shown with marks and tabs are shown with arrows.
showMoveChangeInfo	(+380)	**Byte**	Indicates whether the Move/Change window is active.
moveChangeInfoRect	(+381)	**4 Words**	Standard QuickDraw II Rectangle giving the position of the Move/Change window.
addNewUndo	(+389)	**Byte**	Currently undocumented.
revNum	(+390)	**Word**	Revision number of the version of Medley that created this file. For files created following this standard, use $0100 (for Medley 2.0).
showRulers	(+392)	**Byte**	Indicates whether the rulers are showing.
windowType	(+393)	**Word**	The type of window this is. 0 = document, 1 = clipboard. When creating files, set this to zero.
auxDictPath	(+395)	**129 Bytes**	Class zero GS/OS pathname to the auxiliary dictionary file for this document. When creating files, set this to a null pathname (a length byte of zero).
grayScale	(+524)	**Word**	Whether or not grayScale mode is active. Zero for color, one for grayScale.
printRecordDefined	(+526)	**Word**	Non-zero if a print record is defined. This is used because the Printer and Port drivers must be loaded before calling `PrDefault` or `PrValidate`, and

			that means the boot disk must be on line. If Medley knows the print record is good, it proceeds without calling the Print Manager.
evenPageNumText	(+528)	**48 Bytes**	The text for even-numbered pages to be displayed by the page number.
oddPageNumText	(+576)	**48 Bytes**	The text for odd-numbered pages to be displayed by the page number.
pageNumInfo	(+624)	**32 Words**	A word for each of the pages possible in the file, through the absolute maximum of thirty-two.
affectPageRange	(+688)	**2 Bytes**	The range of pages affected by the Medley "Change Page Numbers" command. The first byte is the beginning page; the second byte is the ending page.
pageNumFont	(+690)	**4 Bytes**	A Font Manager FontID, identifying the font used for the page numbers. This is a two-byte font family number, followed by a one-byte font style and one-byte font size.
startPageNum	(+694)	**Word**	The page number of the first page. This is a zero-based counter; page one is represented as zero.
offsetFromEdge	(+696)	**Word**	The distance in points that the page numbering text appears from the edge of the paper.

Note: The following three fields are in Medley 2.0 files, but do not exist in Medley 1.0 files. If you are reading a 1.0 file, the value of `revNum` will be \$0000. If reading a 1.0 File Object, resize it to 2.0 size (including the three fields below) and initialize their values to the values given below.

maxNumPages	(+698)	**Word**	The maximum number of pages in this document. When creating new files, initialize this to 32 (\$20) unless condensed (below) is non-zero. If condensed is non-zero, you really have to hurt yourself to get this field right. Below is the algorithm Medley uses to calculate this field (in something close but not exactly related to pseudo-code). Please recall that all variables relating to the margins (taken from the file object) are **Fixed**.

```
workHeight := topMrgn + bottomMrgn
if [condensed is non-zero] then workHeight := workHeight * 2
workHeight := pageHeight - workHeight
workHeight := workHeight * [pixels per vertical inch]
workHeight := workHeight + $0000FFFF
[this counts a fractional point as a whole point]
i := HiWord(workHeight) [this gives the integer portion]
i := i + 3 [accounts for 3-pixel page breaks]
i := (16384 - 208) / i [gives number of pages in conceptual drawing
space.  Since Medley allows 48-point characters plus leading, the tallest
a text rectangle may be is 208 pixels.  Text that does not fit in
maxNumPages is kept around in a non-displayable, non-editable, non-
printable page.  Any shortening of the document will cause some or all of
the previously non-displayed text to flow up into the document.]
i := min(32,i) [32 is the absolute maximum number of pages Medley allows
due to QuickDraw's conceptual drawing space limitations.]

maxNumPages := i
```

condensed	(+700)	**Word**	Indicates whether the document is designed to use condensed printing. If non-zero, the document is designed to use condensed printing. When creating files, it is easiest not to deal with condensed printing, so set this field to zero. However, if you wish to create a document that Medley may edit and print as condensed, you must correctly relate this field to the previous one by the algorithm given above.
reserved	(+702)	**6 Bytes**	These six bytes should be set to zero.

The Page Object

Pages are the first-level children of files. There is one page object for each page in a document (file object).

Field	Offset	Size	Description
rect	(+013)	**4 Words**	Standard QuickDraw II rectangle giving the boundary rectangle for this page.
wrapDir	(+021)	**Byte**	The direction of word-wrapping. 1 = Down, 2 = across.
rgn	(+022)	**Long**	Handle to the region for this page in memory. The region is the page rectangle less any areas on that page. Global areas are not subtracted from this region. They are subtracted from the page rectangle for the global page (page zero). On disk, where the page region would be written, you can write a 10, followed by the rectangle for the page. This is a rectangular region. The aggregate size of the page object on disk must include these 10 bytes. This is assuming, of course, that there are no areas on that page to make the page region non-rectangular.
hideGlobalArt	(+026)	**Byte**	A non-zero value indicates that global art is not displayed on this page.
hideGlobalPageParts	(+027)	**Byte**	A non-zero value indicates that global page parts are not displayed on this page.

The Paragraph Object

Paragraphs are the children of the file object; they are **not** the children of page objects since a paragraph may be seen on more than one page or page part. Paragraph objects are, however, stored on disk immediately following page objects and their children. Paragraph objects are first-level objects also.

Field	Offset	Size	Description
wrapHere	(+013)	**Word**	Insertion offset point in paragraph data where wrapping should continue. For wrapping from beginning of paragraph, set this field to zero.
fullWrap	(+015)	**Word**	Same as wrapHere, but indicates at what point miniRect construction or reconstruction must continue. Again, for full-wrapping of a paragraph, set to zero.
rulerOffset	(+017)	**Word**	Offset in bytes from beginning of paragraph object to indicate where the ruler starts. (The ruler is just before the character data, just after the miniRects.) If there is no ruler, then the default ruler is used by Medley. (If the dataOffset value is the same as the rulerOffset, then there is no ruler, and the default ruler will be used.) The default ruler has tabs at each 1/2 inch mark, no indent or paragraph indent, and the right margin is at maximum.
dataOffset	(+019)	**Word**	Offset in bytes from beginning of paragraph object to indicate where character data starts. If there are no miniRects built yet (probable if file is being created outside Medley) and there is no ruler, then this value will be a 32.

numRects	(+021)	**Word**	The number of discrete text rectangles in this paragraph. When creating a file, set fullWrap to zero, and numRects to zero, and place your character data starting at byte 32 of a paragraph object, and the rectangles will be built when the file is loaded by Medley as wrap occurs.
begInvOffset	(+023)	**Word**	Offset from the beginning of the character data where inverse text starts in this paragraph.
endInvOffset	(+025)	**Word**	Offset from the beginning of the character data where inverse text ends in this paragraph.
topLeading	(+027)	**Byte**	The number of pixels leading above each line in this paragraph.
botLeading	(+028)	**Byte**	The number of pixels leading below each line in this paragraphs.
begPgphGap	(+029)	**Byte**	The number of pixels extra leading above this paragraph.
endPgphGap	(+030)	**Byte**	The number of pixels extra leading below this paragraph.
flags	(+031)	**Flag Byte**	Bits 0 and 1 are used to indicate justify mode. 00 = left justify. 01 = right justify. 10 = center justify. 11 = full justify. Bit 7 indicates a page-break after this paragraph.
miniRects	(+032)	**MiniRects**	Any miniRects, if any, are contained here. The number of miniRects is given by numRects above.

MiniRects have the following format:

miniRect.rect	(+000)	**4 Words**	Standard QuickDraw II rectangle that is calculated by the wordWrap routine to bound a line of text.
mr.begOffset	(+008)	**Word**	Offset from start of character data to the first character this miniRect bounds.
mr.endOffset	(+010)	**Word**	Offset from start of character data to just past the last character this miniRect bounds.

A Ruler in the document will be after the miniRects, if there are any. The offset to the Ruler is given by rulerOffset. Rulers are formatted as follows:

leftPgphMrgn	(+000)	**Byte**	The left margin for this paragraph, in sixteenths of an inch.
rightPgphMrgn	(+001)	**Byte**	The right margin for this paragraph, in sixteenths of an inch. This is an offset from the default right margin from Medley's "Set margins" command. For example, the value 16 represents a right margin one inch to the left of the default right margin.
pgphIndent	(+002)	**Byte**	The indentation for this paragraph, in sixteenths of an inch.
numTabs	(+003)	**Byte**	The number of tabs in this ruler.
tabs	(+004)	**Tabs**	There are numTabs of these.

Tabs are formatted as follows:

tab		**Flag Word**	Tabs consist of a high byte of flags and low byte of position. The bits are assigned as follows: Bits 15-12 = Reserved; set to zero. Bits 11-10 = Tab Leader style:

	00 = No leader 01 = Leader of dots (.......) 10 = Leader of dashes (- - - - - - -) 11 = Solid Leader (____________)
Bits 9-8 =	Tab Type: 00 = Left Tab 01 = Right Tab 10 = Center Tab 11 = Decimal Tab
Bits 7-0 =	**Byte** value; the position of this tab as an offset from the left margin in sixteenths of an inch. A value of sixteen indicates a tab one inch to the right of the left margin.

Following miniRects and rulers is the actual character data for this paragraph. This is all **Bytes**. However, a **Byte** value of $01 through $07 indicates the beginning of a **Font Escape**. Font Escapes indicate changes in style or size of the text, and are formatted as follows:

FontEscape	(+000)	**Byte**	An indication of the type of text the following fontID affects: 1 = Regular Text 2 = Superscript Text 3 = Subscript Text 4-7 = Reserved; do not use
fontID	(+001)	**4 Bytes**	A Font Manager FontID, identifying the font used for the page numbers. This is a two-byte font family number, followed by a one-byte font style and one-byte font size.

The text portion of a paragraph **always** begins with a Font Escape and ends with the end-of-paragraph character **Byte** $A6 (¶). This makes the minimum size of a paragraph (assuming no miniRects or rulers) thirty-eight bytes (32 bytes for the Paragraph Object, five bytes for the Font Escape and one byte for the $A6).

The Area Object

Area Objects are the children of pages or paragraphs.

Field	Offset	Size	Description
type	(+013)	**Byte**	The type of area this area object describes. Possible values are: 0 = Null Area 1 = Group Area 2 = Rectangular Area 3 = Round Rectangular Area 4 = Oval Area 5 = Polygon Area
select	(+014)	**Byte**	This value is one if this area is selected.
showBorder	(+015)	**Byte**	This value is one if the border of this area is showing.
contentType	(+016)	**Byte**	0 = Art, 1 = Wrap Down, 2 = Wrap Across.
rgn	(+017)	**Long**	Handle to the region that describes the shape of this area. On disk, this region is at the end of this object (see the Reading The File section of this Note).
interiorRgn	(+021)	**Long**	Handle to the regions that describes the interior of this area. On disk, this region is at the end of this object (see the Reading The File section of this Note).
sizingRgn	(+025)	**Long**	Handle that contains all the sizing dots. It is too slow to draw them one at a time. Also, detecting that the user clicked in a sizing dot can be done quickly -- just not which one.
flags	(+029)	**Word**	Only bit zero of this word is significant; if set it indicates this area should be printed to LaserWriters in gray-scale. All other bits of this word should be zero.
reserved	(+031)	**Word**	Reserved for Milliken. Set to zero.

At this point is the description of the area itself. This description varies on the type field above:

For rectangles (type = 2):

Field	Offset	Size	Description
rect	(+033)	**4 Words**	Standard QuickDraw II rectangle describing the rectangle for this area.

For round rectangle (type = 3):

Field	Offset	Size	Description
rect	(+033)	**4 Words**	Standard QuickDraw II rectangle describing the boundary rectangle for the round rect.
height	(+041)	**Word**	The height of the oval portion of the rectangle.
width	(+043)	**Word**	The width of the oval portion of the rectangle.

For ovals (type = 4):

Field	Offset	Size	Description
oval	(+033)	**4 Words**	A standard QuickDraw II rectangle describing the bounding rectangle for this oval. The oval drawn is the ellipse inscribed in this rectangle.

For Polygons (type = 5):

Field	Offset	Size	Description
polygon	(+033)	**Bytes**	A handle to a QuickDraw II polygon. This handle may be passed to QD Polygon routines. On disk, this

			polygon is appended to the end of this object (see the Reading The File section on this Note).

These objects are the last items in the area object.

The Art Object

Art Objects are the children of pages, paragraphs or areas.

BBox	(+013)	**4 Words**	A standard QuickDraw II rectangle representing the bounding box of this art object.
offsetFromRgn	(+021)	**2 Words**	Normally zero. The area containing an art image can be grown and shrunk. The art within it is not clipped to the bounding rectangle of the area until the user deselects the area. (If it is saved to disk while selected, then it is saved unclipped). This allows the user to experiment with different shapes without clipping the drawing within. If the drawing is to the left of the left edge of the area, or is above the top edge, then this offset indicates by how much.
artImage	(+025)	**Bytes**	The actual bitmap of the art image.

The Document Dictionary Object

The Document Dictionary Object is the very last child of the file object, and contains all the words the spelling checker should ignore even though they are not in the main dictionary.

count	(+013)	**Word**	The number of word entries in this dictionary object.
wordList	(+015)	**Word Entries**	List of dictionary word entries.

The format of word entries is as follows:

recordLength	(+000)	**Byte**	The length of this record.
replaceFlag	(+001)	**Byte**	Reserved, set to zero.
newWord	(+002)	**C String**	The word in question. This word should be counted as spelled correctly, and is not in the Main or Auxiliary Dictionary.

The length of a record is the length of string plus three bytes (one for recordLength, one for replaceFlag, and a zero termination byte).

Reading the File

When reading a Medley file, objects with regions or polygons will have to be treated specially, since the handles in the objects are invalid and the regions or polygons actually follow the object in the disk file.

A sequence for reconstructing Medley files in memory is as follows:

1. Open the file, or set the mark to zero on an open file.

2. Start with a handle that is 13 ($0D) bytes long. Pass this handle to the routine starting in step three.
3. Save the handle passed to this routine, and read four bytes from disk. This Long is the total size of an object, including any regions or polygons appended.
4. Read the 13-byte object header into the handle passed to this routine. The `endData` field of the header gives the size of the object, minus any associated regions or polygons. Resize the object's handle (the handle passed to you) to this size.
5. Read the rest of the object (`endData` - 13 bytes) into the object's handle.
6. Save the value of `numChildren` in a local variable and set the `numChildren` field in the object header to zero. The field in the header represents the number of children read from disk; setting this to zero properly indicates that you haven't read any of the children yet.
7. Look at the object type field. If the object is a file, area, or page object, it may have a region associated with it. If the object is an area object, it may also have a polygon associated with it (if the area type field indicates this is so). You can tell if the object has any appended structures by comparing the total object size (read in step three) with the `endData` field (read in step four); if an object has no appended structures, the two values will be the same.

   If there are structures appended to the object, first zero all the handles to the regions inside the object. This allows elegant error recovery if an error occurs while reading the region or polygon. When the handles are zeroed, read the next two bytes from the disk. This Word is the size of the region or polygon in bytes. Create a handle of that size, place it in the object's field for this handle, and place the size Word in the first two bytes of the new handle. Now read the object from disk into the new handle starting at the beginning `+002` (past the size Word).

   Continue in this fashion until all appended regions or polygons have been read from disk. Any appended structures will be stored in the same order as their handles occur in the object.

   Note: By zeroing the handles before reading the objects, you can return from this function with an error, and the calling routine will be able to dispose of all handles that were actually created. The calling routine will know if a handle was actually created or not by examining the handle field in the object; `NIL` handles were not created.

8. Execute a loop for the old number of children (0 to `oldNumChildren`-1):
9. Create an object that is 13 ($0D) bytes long. Add this handle to the end of the parent object that was last read. Increment the number of children. You have just added a child into the child table for an object.
10. Call the recursive subroutine beginning in step three, passing it the handle you just created. If it returns an error, return the error. This gets you out of the recursion with the correct error, no matter how many levels deep you are.

11. Keep looping until out of children to read. The EOF condition does not have to be checked, since you will run out of children when you reach the end of the file. If an EOF is reached before you read all the children, you did something wrong.
12. Return no error—the file was successfully read.
13. When done with all this, you will return to the code just beyond step two, where you first called the recursive subroutine at step three. If an error is returned, dispose of all the handles created by the recursive function. Even if the file read is aborted, the tree is complete for as much as was read. (This is why the `numChildren` field is incremented as you read the file.) An alternate way to handle this is to use a different `userID` for the handles created when reading the file; this allows you to dispose of all of them with one `DisposeAll` call.
14. Close the file if you opened it, or reset the mark to its previous position if it was already open.

The entire file does not have to be read from disk. By using the size field, you can skip to the next object in the file. Using this technique, you can scan the file for whatever it is that interests you.

Note: You may have noticed that objects successfully created in memory will have a table of handles to children at the end. Objects on disk will not have these handles, since the handles on disk are meaningless. The child handle table is reconstructed as the file is loaded into memory.

Object Ordering

The file object is the first you will encounter in a Medley file. Its children are ordered as follows:

Page Object—Page #0. This is the global page object, containing all global areas.
Page Object—Page #1. This is the page object for page #1; it must exist.

Other objects are optional, but will appear in the following order:

Page #2 through Page #n
Paragraph #1 through Paragraph #n
Dictionary Object

Some Example Structures

Medley was written mostly in C. Below are some structures relevant to C programs reading Medley files. Descriptions of the fields may be found earlier in this Note.

```
#define NULLOBJ      0      /*  Object type assignments.  */
#define ROOTOBJ      1      /*  These are used in the deskObj 'types' field.  */
#define FILEOBJ      2
```

```
#define PAGEOBJ      3
#define PGPHOBJ      4
#define AREAOBJ      5
#define ARTOBJ       6
#define DOCDICTOBJ   10

#define AREANULL     0      /*  Area object sub-type assignments.  */
#define AREAGROUP    1      /*  These are used in the areaObj 'types' field.  */
#define AREARECT     2
#define AREARRECT    3
#define AREAOVAL     4
#define AREAPOLY     5


#define medleyMainType      0x54
#define medleyAuxType       0xDD3E
#define medleyInfo          1
#define auxDictType         2


#define ARTCONTENT   0      /* These are used in the 'contentType' field of area
objects.*/
#define WWDOWN       1
#define WWACROSS     2
#define LWGRAYSCALE  0x0001


#define SAMEESC      0      /*  These are used in paragraph objects.  */
#define FONTESC      1
#define SUPERESC     2
#define SUBESC       3
#define ESCAPES      7


#define SIZEFONTESC  5      /*  More paragraph equates.  */
#define ENDPGPHCHR   0xA6
#define TABCHR       9
#define SOFTHYPHEN   30
#define STICKYSPACE  31


#define PAGEBREAK    0x80   /*  These are used in the pgphObj 'flags' field.  */
#define LEFTJUST     0x00
#define RIGHTJUST    0x01
#define CENTERJUST   0x02
#define FULLJUST     0x03
#define JUSTTYPES    0x03

#define LEFTTAB      0x00   /*  These are used in the ruler field of paragraph objects.
*/
#define RIGHTTAB     0x01
#define CENTERTAB    0x02
#define DECIMALTAB   0x03
#define TABTYPES     0x03
#define NOLEADER     0x00
#define DOTSLEADER   0x01
#define DASHESLEADER        0x02
#define SOLIDLEADER  0x03

typedef struct Ruler {
       unsigned char leftPgphMrgn;
       unsigned char rightPgphMrgn;
       unsigned char pgphIndent;
       unsigned char numTabs;
       unsigned int  tab[];
} Ruler;
```

```
#define NEWREVNUM 0x0100


typedef union URect {
       Rect    rect;
       struct {
              long   p1;
              long   p2;
       } point;
       struct {
              Point  p1;
              Point  p2;
       } ele;
} URect;

typedef union UPoint {
       Point  ele;
       long   point;
} UPoint;

typedef struct region {
       unsigned int  size;
       union URect   BBox;
       int           data[];
} region;

typedef struct polygon {
       int           size;
       union URect   BBox;
       union UPoint  point[];
} polygon;
```

```
typedef union ourFontID {
      unsigned long fid;
      struct {
            unsigned int  famNum;
            char          fontStyle;
            char          fontSize;
      } f;
} ourFontID;

struct deskObj {
      char          type;               $00
      unsigned int  numChildren;        $01
      unsigned long endData;            $03
      unsigned long reserved;           $07
      unsigned int  objRefNum;          $0B
                                        $0D

      union d {
            data[];             /*  Plain label object access field.  */

            struct file {               /*  Level 1 objects are files.  */
                  union URect   rect;                                  /* $0D */
                  char          pathName[129];                         /* $15 */
                  char          saved;                                 /* $96 */
                  GrafPortPtr   windowPtr;                             /* $97 */
                  char          windowNameIndx;                        /* $9B */
                  long          windowOrigin;                          /* $9C */
                  long          windowSize;                            /* $A0 */
                  long          COrigin;                               /* $A4 */
                  struct deskObj      **editHndl;                      /* $A8 */
                  unsigned int  editOffset;                            /* $AC */
                  union URect   cursor;                                /* $AE */
                  char          showAllBorders;                        /* $B6 */
                  union URect   updateRect;                            /* $B7 */
                  unsigned long topMrgn;                               /* $BF */
                  unsigned long bottomMrgn;                            /* $C3 */
                  unsigned long leftMrgn;                              /* $C7 */
                  unsigned long rightMrgn;                             /* $CB */
                  unsigned long gutterMrgn;                            /* $CF */
                  unsigned long pageWidth;                             /* $D3 */
                  unsigned long pageHeight;                            /* $D7 */
                  int           selectPage;                            /* $DB */
                  int           numSelected;                           /* $DD */
                  int           sizingDot;                             /* $DF */
                  int           effectivePage;                         /* $E1 */
                  PrRec         printRecord;                           /* $E3 */
                  int           interruptMode;                         /* $16F */
                  char          editScroll;                            /* $171 */
                  struct deskObj **firstHndl;                                 /* $172 */
                  int           firstMrn;                              /* $176 */
                  unsigned int  selectMode;                            /* $178 */
                  char          showPgphMarks;                         /* $17A */
                  char          showSpaces;                            /* $17B */
                  char          showMoveChangeInfo;                    /* $17C */
                  union URect   moveChangeInfoRect;                    /* $17D */
                  char          addNewUndo;                            /* $185 */
                  unsigned int  revNum;                                /* $186 */
                  char          showRulers;                            /* $188 */
                  unsigned int  windowType;                            /* $189 */
                  char          auxDictPathname[129];                  /* $18B */
                  unsigned int  grayScale;                             /* $20C */
                  unsigned int  printRecordDefined;                    /* $20E */
                  char          evenPageNumText[48];                          /* $210 */
                  char          oddPageNumText[48];                    /* $240 */
                  unsigned int  pageNumInfo[MAXNUMPAGES];              /* $270 */
                  unsigned int  affectPageRange;                       /* $2B0 */
```

```
        ourFontID     pageNumFont;                                  /* $2B2 */
        unsigned int  startPageNum;                                 /* $2B6 */
        unsigned int  offsetFromEdge;                               /* $2B8 */
        unsigned int  maxNumPages;                                  /* $2BA */
        unsigned int  condensed;                                    /* $2BC */
        char          reserved[6];                                  /* $2BE */
} file;                                                             /* $2C4 */

struct page {
        union URect   rect;                                         /* $0D */
        char          wrapDir;                                      /* $15 */
        region        **rgn;                                        /* $16 */
        char          hideGlobalArt;                                /* $1A */
        char          hideGlobalPageParts;                                 /* $1B */
} page;                                                             /* $1C */

struct pgph {               /*  Must be level 2 or greater.  */
        unsigned int  wrapHere;                                     /* $0D */
        unsigned int  fullWrap;                                     /* $0F */
        unsigned int  rulerOffset;                                  /* $11 */
        unsigned int  dataOffset;                                   /* $13 */
        unsigned int  numRects;                                     /* $15 */
        unsigned int  begInvOffset;                                 /* $17 */
        unsigned int  endInvOffset;                                 /* $19 */
        char          topLeading;                                   /* $1B */
        char          botLeading;                                   /* $1C */
        char          begPgphGap;                                   /* $1D */
        char          endPgphGap;                                   /* $1E */
        char          flags;                                        /* $1F */
        struct miniRect {                                           /* $20 */
                union URect   rect;
                unsigned int  begOffset;
                unsigned int  endOffset;
        } miniRect[];
             /*  Ruler goes here if there is a custom ruler for this
                 paragraph.*/
             /*  Text starts after the ruler.  Text always starts
                 with a fontEsc.  A fontEsc is 5 bytes, a typeByte
                 followed by the fontID.  Text always ends with end-
                 of-pgph chr. */
} pgph;                                                             /* $20 */

struct area {
        char          type;                                         /* $0D */
        char          select;                                       /* $0E */
        char          showBorder;                                   /* $0F */
        char          contentType;                                  /* $10 */
        region        **rgn;                                        /* $11 */
        region        **interiorRgn;                                /* $15 */
        region        **sizingRgn;                                  /* $19 */
        unsigned int  flags;                                        /* $1D */
        unsigned int  reserved;                                     /* $1F */
        union obj {
                union  URect  rect;                                 /* $21 */
                struct rrect {
                        union URect   rect;                         /* $21 */
                        int           height;                       /* $29 */
                        int           width;                        /* $2B */
                } rrect;
                union  URect  oval;                                 /* $21 */
                polygon       **poly;                               /* $21 */
        } obj;
} area;                                                             /* $2D */
```

```
            struct art {
                  union URect   BBox;                                  /* $0D */
                  union UPoint  offsetFromRgn;                         /* $15 */
                  char          artImage[];                            /* $19 */
            } art;

            struct docDict {
                  unsigned int  count;                                 /* $0D */
                  char          wordList[]                             /* $0F */
            } docDict;                                                 /* $0F */

      } d;
};
```