MARPLOT® Technical Documentation 3.2.3

Contents

MARPLOT Simple Point Format

MARPLOT can import and export point (symbol) objects in a tab-delimited text file, where each line, one per object, has the following format (<> represents a tab character):

longitude <> latitude <> name <> layer name <> map name <> symbol <> color <> ID

Only the first two fields (longitude and latitude) are required. The remaining fields are optional; you can specify as many as you want. However, if you want to specify a field, you must also include all fields to its left on the line. For instance, if you want to specify the symbol, you must also include name, layer, and map, but including color and ID is still optional. For further flexibility in cases where you want to specify a field to the right of an unspecified field, you can use a placeholder for the unspecified field, which causes MARPLOT to use its default value. In the example just given, suppose you wanted to specify the symbol, but not the layer or map. In this case you would use the placeholder value 0 (zero) for both the layer name and map name fields.

The following table shows, for each field, the type of data in the field, the default value used if the field is not present or is equal to the placeholder value, and the placeholder value to force MARPLOT to assign the default.

field	format / example	default value	placeholder value
longitude	decimal degrees (west negative) / -122.123456	N/A	N/A
latitude	decimal degrees (south negative) / 47.123456	N/A	N/A
name	string of characters / Observation Site	object's ID number	0
layer name	string of characters / My Sites	the name of the topmost layer	0
map name	string of characters / CAMEO Map	User's Map	0
symbol	either the symbol LANDMARK (specifying the standard flag symbol) or the ASCII value of the desired symbol in the MARPLOT font	LANDMARK	0
color	one of the names of the available colors (as listed in MARPLOT Color menu)	BLACK	0
ID	a 16-digit number (the digits are actually hexadecimal so you can use A-F in addition to 0-9) NOTE: it is important that any IDs you assign are unique on a given layer of a given map (i.e., no two objects on a given layer of a given map should have the same ID)	a random code	N/A

GENERATE Format

This format is designed to be compatible with ArcInfo's GENERATE command. This provides a mechanism for transferring data between MARPLOT and ArcInfo, or any other program that can read and write in the GENERATE format. It provides an intermediate option between the simplicity of the MARPLOT Simple Point Format and the complexity of the MIE format. In contrast to the Simple Point Format, it provides a mechanism for transferring polygons and polylines.

The basic coodinate information in the GENERATE format is kept in one of three types of text (TXT) files, depending on the type of the objects in the file. For point objects, each line of the file contains a longitude/ latitude coordinate pair, preceded by an index number. For polygon objects, the file contains a sequence of polygons, where each polygon is a sequence of lines. The first line of a polygon is a longitude/ latitude coordinate pair, preceded by an index number. This first coordinate pair specifies a center or labelling point for the polygon (this point is not used by MARPLOT). The remaining lines of a polygon are longitude/ latitude coordinate pairs, without an index number. The final line of a polygon is the keyword END. For polyline objects, the file contains a sequence of polylines, where each polyline is treated the same as a polygon in a polygon file, except that the first line of each polyline contains only an index number (and not the coordinates of a labelling point). In all cases, the file ends with the keyword END. Below are the sample GENERATE TXT files, one for each type of object. Note that the index numbers are arbitrary: the index numbers in a given file must be unique, but need not be sequential and need not start with 1.

sample file of point objects: POINTS1.TXT

```
1 -77.518536 38.782300
2 -77.415016 38.774448
END
```

sample file of polygon objects: POLYS1.TXT

```
5 -77.595456 38.679660
-77.595456 38.679660
-77.565984 38.640400
-77.595456 38.679660
END
6 -77.373320 38.813708
-77.373320 38.813708
-77.379784 38.771644
-77.358224 38.735188
-77.339528 38.822120
-77.362536 38.803612
-77.373320 38.813708
END
END
```

sample file of polyline objects: LINES1.TXT

```
3
-77.536512 38.764912
-77.502000 38.743600
-77.543696 38.729016
-77.495528 38.724528
-77.478280 38.693684
END
4
-77.464616 38.768840
-77.466056 38.743040
-77.398480 38.747528
-77.382664 38.720044
END
END
```

Numbers on a line can be delimted by spaces, tabs, or commas, or any combination of these.

MARPLOT's GENERATE format extends these three basic TXT file formats with the option for two additional files, corresponding to each basic TXT file, which specify attributes of each object in the TXT file. The two files are the data (DAT) file and the field (FLD) file. When MARPLOT imports a GENERATE TXT file, it looks for a file in the same folder as the TXT file, with the same name, except with a ".DAT" extension in place of the ".TXT" extension. If a DAT file is present, it contains attributes for the objects in the TXT file. The possible attributes are the same as those in the Simple Point Format: object name, layer name, map name, symbol, color, ID. By default, MARPLOT assumes that all of these fields are present in the DAT file, in the order just given, for each object in the TXT file. The FLD file (a file with the same name as the TXT file but with a ".FLD" extension)

can be used to override this default in order to specify only some of the fields and/ or to change their order. A FLD file contains just a single line of text. The line contains one or more of the following keywords, separated by spaces and/ or commas: INDEX, NAME, LAYER, MAP, SYMBOL, COLOR, ID. The keywords specify which fields are present in the corresponding DAT file, and what their order is on each line. A special case is the INDEX keyword, which must alway appear, and must always be the first item on the line. This is because the object's index number in the TXT file must also appear as the index number of the corresponding line in the DAT file.

The following example shows the POINTS1.TXT file from above, along with corresponding DAT and FLD files. In this case, just the color, symbol, and object name are specified.
(**Note**: an extended format is available for RGB colors; see MARPLOT colors)

POINTS1.TXT	POINTS1.DAT	POINTS1.FLD

`1 -77.518536 38.782300`
`2 -77.415016 38.774448`
`END`
	`1 BLUE 53 SITE_A;_CA`
	`2 GREEN 51 SITE_B;_WA`
	`END`
		`INDEX COLOR SYMBOL NAME`

Because spaces and commas are used as delimiters between fields in the DAT file (the same as with the TXT file), spaces and commas may not be used within an object name, layer name or map name in the DAT file. However, MARPLOT allows spaces and commas in names. To get around this problem, MARPLOT automatically converts underscores to spacesand semicolons to commas when importing, and does the reverse conversions when exporting. Thus, the name "`SITE_A;_CA`" in the file above actually represents "`SITE A, CA`" in MARPLOT.

When MARPLOT exports a set of GENERATE TXT files, it writes corresponding DAT and FLD files for each TXT file. The fields included in the FLD and DAT files are chosen by the user at the time of the export. MARPLOT always writes a total of nine files for a GENERATE export, one of each type of file (TXT, DAT, FLD) for each type of object (point, polygon, polyline). It does this even if not all three types of objects are being exported. For instance, if you export just points, you will still get TXT, DAT, and FLD files for polygons and polylines, but they will be empty.

MARPLOT import/export (MIE) file format

```
marplot-object = keys head body

keys = owner modifier location mod-date

owner = modifier = "4 (or fewer) character string"

location-code = "5 (or fewer) character string"

mod-date = "mm/dd/yyyy"

head = version-number prefix name alias-count layer map type id
        digitization-scale CFCC FIPS-place-code etc state-county
```

(**Note**: In head of `ALIAS` objects, all fields are same as original object, except `id` is prefixed with unique digit(s) and `prefix`, `name` differ.)

```
alias-count = version-number = short

layer = prefix = name = "string"
map = "string" | "" (if empty, current "user's map" is used)

type = RECT | CIRCLE | POINT (SYMBOL) | POLYLINE | POLYGON | TEXT |
        PICTURE | ALIAS

id = "16-digit hex string" | "" (if empty, random ID is assigned)

digitization-scale = FIPS-place-code = state-county = long

etc = ONLY | ETC

CFCC = "XXX" (3-character string)
       -> for city/place polygon objects, CFCC is M00
       -> for county polygon objects, CFCC is M01
       -> for census block polygon objects, CFCC is M02
       -> for PICT objects, CFCC starts at X00 (unclassified)
       -> for TEXT objects, CFCC starts at X00 (unclassified)

body = color line-width symbol long lat ;;; for POINT

        | color line-width line-pat fill-pat
          lo-long low-lat hi-long hi-lat ;;; for RECT, CIRCLE

        | frame "filename" lo-long low-lat hi-long hi-lat ;;; for PICT

        | color frame font style "text"
          lo-long low-lat hi-long hi-lat ;;; for TEXT

        | color line-width line-pat
          fill-pat { segment ... segment } ; for POLYLINE, POLYGON

        | id ; for ALIAS
```

```
lat = long = low-lat = lo-long = hi-lat = hi-long = signed float
       | floatdirection | deg°min'sec"direction | signdeg°min'sec"

direction = N | S | W | E

sign = + | -  (Note: western longitudes are -, eastern longitudes are +.)

color = BLACK | WHITE | DARKGRAY = DARKBLUE | GRAY | LIGHTGRAY | BROWN |
         LIGHTBROWN  = OLIVE | DARKGREEN | GREEN | LIGHTBLUE | BLUE |
         PURPLE | PINK | RED | ORANGE = AQUA | YELLOW
         (Note: an extended format is available for RGB colors; see MARPLOT colors)

font = style = symbol = integer (short)

frame = YES | NO

line-width = 1 | 2 | 4 | 6 | 8 | 10

fill-pat = BLACK | WHITE (NONE) | DARKGRAY | GRAY | LIGHTGRAY
            | VERTSTRIPES | HORIZSTRIPES | UPSTRIPES | DOWNSTRIPES
            | BOXES

line-pat = BLACK | WHITE (NONE) | DARKGRAY = TWOPOINT
            | GRAY = THREEPOINT | LIGHTGRAY = FOURPOINT
            | VERTSTRIPES = DOTS | HORIZSTRIPES = DASHDOTDOT
            | UPSTRIPES = DASHDOT | DOWNSTRIPES = DASHES | RAILROAD

segment = { from-to long lat { attribute value } ... { attribute value } }

from-to = FROM | TO

attribute = TLID | CFCC | VERS | SAL | SAR | EAL | EAR | ZCL | ZCR | INVIS

value = integer (long)
```

Meaning of terms

`owner, modifier` = code of person/ organization that created/ modified object.
`location-code` = usually FIPS state/ county code of county the object is in.
`version-number` = version of MIE syntax used (current version is 2).
`prefix` = the prefix of the object's name (usually for roads) such as "N" or "SW".
`alias-count` = the number of alias "objects" (alternate) names the object has.
`digitization-scale` = scale at which object was originally digitized.
`CFCC` = Census Feature Classification Code.
`FIPS-place-code` = city/ town FIPS number, unique within the given county.
`etc` = `ETC` if object crosses into other places (cities/ towns), otherwise `ONLY`.
`state-county` = usually the same as the `location-code`.

`TLID` = TIGER/ Line ID number of segment from TIGER record type 1.
`VERS` = TIGER/ Line database version number.
`SAL, SAR, EAL, EAR` = start address left/ right, end address left/ right
`ZCL, ZCR` = ZIP code left/ right
`INVIS` = segment is invisible (`value` = 0).

Sample MIE Entries

The text in the box represents an actual sample MIE file, except that the <bracketed> terms would have to be filled in or left empty. This sample file contains two objects, a polygon and a point.

```
"FRED" "FRED" "00000" "05/24/1994" 2
"" "Central Park" 0 "layer name" "<map name>" POLYGON "<id number>" 0 "X00" 0 ONLY
0
BLACK 1 BLACK NONE { { FROM -76.992700 38.844000 }
                     { TO -76.992400 38.842600 }
                     { TO -76.992100 38.841100 }
                     { TO -76.992200 38.840000 }
                     { TO -76.991700 38.839200 } }

"MSIS" "MSIS" "00000" "05/24/1994" 2
"" "ABC Chemical" 0 "layer name" "<map name>" POINT "<id number>" 0 "X00" 0 ONLY 0
BLACK 1 LANDMARK -76.992700 38.844000
```

Note: Items in <brackets> can be empty (i.e., "").

Note: The constant LANDMARK can be replaced with any integer ascii value to specify any character in the MARPLOT font.

Special MIE Symbols

ASCII code	Macintosh character	Windows character	MIE value
0x27	single quote	single quote	minutes
0x22	double quote	double quote	seconds
0xA1	degrees	i (monetary unit)	degrees
0xB0	infinity sign	degrees	degrees
0xBA	integral sign	degrees	degrees
0xAB	slanted apostrophe	<< (Romance quote)	minutes
0xD5	smart apostrophe	O with tilde	minutes
0xB4	yen	slanted apostrophe	minutes
0x92	accented i	smart apostrophe	minutes
0xD3	smart close quotes	O with accent	seconds
0x94	i with caret	smart close quotes	seconds

MARPLOT 1.0.1 import/export file format

(**Note**: In MARPLOT 3.0, this format is used for the IMPT IAC message. Also, MARPLOT 3.0 can import files in this format (useful for transferring objects from MARPLOT 1.0.1 to MARPLOT 3.0. For all other purposes, use the new MIE format.)

The first line of the file contains just the number 1. This is a flag for MARPLOT that the file is in the extended format.

The next lines, which are optional, associate overlay names with overlay ID numbers. For each line, the format, including the leading asterisk, is: * <overlay ID> <overlay name>

Then, on the subsequent lines, each object is described by a group of eight or nine lines:

first line (all on the same line even though there are three lines here):
<object ID>, <overlay ID>, <type code>, <hilat>, <hilong>, <lowlat>, <lowlong>, <color>, <font>, <size>, <style>, <fill pattern>, <line pattern>, <line width>, <symbol code or 0>, <object name>

next line (for polygons only):
[indent] <number of points>, <point1 lat>, <point1 long>, ... , <pointn lat>, <pointn long>

next lines:
[indent] <pseudo signature>
[indent] <real signature>
[indent] <alias>
[indent] <application path>
[indent] <document path>
[indent] <record>
[indent] <note>

Notes:

1) Each object is defined by 8 lines (9 for polygons). There are no blank lines between objects; the ID of one object starts the line right after the note of the previous object.
2) [indent] indicates that the line must start with at least one space or tab character. Other lines must NOT be indented.
3) If you want to leave one or more of the last 7 lines of a given object's definition blank, you must still indent the line (a space followed immediately by a return would do).
4) Here is a description of each of the fields:
   (a) The <object ID> is a 16-character hexadecimal string that MARPLOT uses to uniquely identify objects. You should never invent object ID's yourself. When importing a new object, use -1 for <object ID>. This is a flag to MARPLOT to generate a new ID for this new object. You should fill in the <object ID> with a real id only when you want to modify an object that you know is on the map. In this case, MARPLOT will not create a new object but will modify the attributes of the object whose id you specify. If you export objects from one MARPLOT map and import them onto another, the objects will retain their IDs. Note, therefore, that there are two ways in which an object can be imported. A "new" import creates a new object with a new id. An "overwrite" import modifies an object that already exists on the map. This new/ overwrite terminology is used below.

(b) The <overlay ID> is a small (base 10) integer indicating which overlay the object is on. Each map contains at least one overlay, which has some ID number. If the <overlay ID> number for an object being imported is the ID of an actual overlay on the map, the object will be placed on that overlay. Otherwise, the object will be placed on the top overlay. Since -1 is not a valid overlay number, you can use -1 for <overlay ID> to put the object on the top overlay, for a new import, or retain the object's previous overlay, for an overwrite import. If overlays have been defined on starred lines at the top of the file, these can override the normal behavior. In particular, if the <overlay ID> field of the object being imported matches one of the <overlay ID> numbers at the top of the import file, the object will be placed on the overlay with the name that is associated with this ID at the top of the file. If there is no overlay with this name, a new overlay with this name is created. Thus, objects "carry along" their overlays when they are exported from one map and imported onto another.

(c) The <type code> is a small integer that detemines the type of the object. The type codes are: line = 0, ellipse = 1, rectangle = 2, symbol = 5, polygon = 6. You can use -1 to retain the object's type in an overwrite import. For new imports, you must provide a type. On an overwrite import, it is an error to specify a type other than the one the object previously had.

(d) The <hilat>, <hilong>, <lowlat> and <lowlong> fields specify the bounding rectangle of the object. These are decimal numbers. The "hi" values are nothern- and western-most, and the "low" values are southern and eastern-most. Negative values indicate southern and eastern hemispheres. For point objects, <hilat> = <lowlat> and <hilong> = <lowlong>. For polygon and ellipse objects, <hilat> = highest latitude value, <lowlat> = lowest latitude value, etc. These fields can be -1 to keep the object in the same position for an overwrite import. For a new import, you must provide real lat/ long values.

(e) The <color> is a small integer from 1 to 16 indicating the color of the object. The colors are ordered in the same way as the Color menu in MARPLOT:

black = 1, white = 2, dark grey = 3, grey = 4, off white = 5, brown = 6,
light brown = 7, dark green = 8, green = 9, light blue = 10, blue = 11,
purple = 12, pink = 13, red = 14, orange = 15, yellow = 16.

For an overwrite import, use -1 to retain the object's previous color. On a new import, -1 can be used to give the object the default color for its overlay.

(f) The <font> is the number of the font in which the object's name is drawn. 0 = Chicago, 3 = Geneva. Other font numbers can be found on page I-236 of Inside Macintosh. For an overwrite import, use -1 to retain the object's previous font. On a new import, -1 can be used to give the object the default font for its overlay.

(g) The <size> is the size in which the object's name is drawn. For an overwrite import, use -1 to retain the object's previous size. On a new import, -1 can be used to give the object the default size for its overlay.

(h) The <style> is the style in which the object's name is drawn. 0 = plain. Other style numbers are generated by adding together constants as described on page I-152 of Inside Macintosh. For an overwrite import, use -1 to retain the object's previous style. On a new import, -1 can be used to give the object the default style for its overlay.

(i) The <fill pattern> determines the pattern with which the object will be filled.

0 = no fill, 1 = black, 2 = white, 3 = dark grey, 4 = gray, 5 = light gray,
6 = vertical stripes, 7 = horizonal stripes, 8 = downward-sloping stripes,
9 = upward-sloping stripes.

Fill pattern has no meaning for symbol and line objects but you must still provide a value. For an overwrite import, use -1 to retain the object's previous fill pattern. On a new import, -1 can be used to give the object the default fill pattern for its overlay.

(j) The <line pattern> determines how the outline of the object will be drawn.

1 = black, 2 = white, 3 = dark grey, 4 = gray, 5 = light gray,
6 = vertical stripes, 7 = horizonal stripes, 8 = downward-sloping stripes,
9 = upward-sloping stripes, 10 = plaid.

Line pattern has no meaning for symbol objects but you must still provide a value. For an overwrite import, use -1 to retain the object's previous line pattern. On a new import, -1 can be used to give the object the default line pattern for its overlay.

(k) The <line width> is a small integer that determines the width (and height) of the outline of the object. Line width must be one of 1, 2, 4, 6, 8 or 10. For symbol objects, <line width> determines the size of the dots when symbols are shown as dots in MARPLOT. For an overwrite import, use -1 to retain the object's previous line width. On a new import, -1 can be used to give the object the default line width for its overlay.

(l) The <symbol code or 0> field should be 0 for all objects except symbol objects. For symbols, this field is the ascii number of the character in the Marplot font that is drawn to represent the object. You can determine these ascii numbers by looking at the Marplot font using the ResEdit utility program. Alternatively, you can figure them out by looking at the symbol menu in MARPLOT, which shows all of the characters in the Marplot font, five per line, starting at ascii 0. For an overwrite import, use -1 to retain the object's previous icon. On a new import, -1 can be used to give the object the default icon for its overlay.

(m) The <object name>, which ends the line, is any string of characters. Only the first 30 characters are used by MARPLOT in the object's name. For an overwrite import, use -1 to retain the object's previous name. On a new import, you must provide a name (which can of course be empty).

(n) If the type code of an object is 6, the object is a polygon, and MARPLOT expects that the second line of the object's definition will contain the points that define the polygon. The first number on this line indicates the number of lat/ long pairs to follow. Then come the pairs, again using decimals with western longitudes positive. Even if there are very many points, they must all be on this one line of the file. To indicate that the polygon is "closed", the coordiates of the first point should be the same as the coordiates of the last point. Otherwise the polygon is considered "open". For an overwrite import, if you have specified the <type> with -1, but the object is a polygon, you must NOT include this ine containing the polygon points. On the other hand, if you do specify <type> 6, you MUST include this line.

The next six lines of the object definition are used to define link information between the object and an external application.

(o) The <pseudo signature> is a four character identifier, which is the pseudo signature of the application to which the object is linked. Pseudo signatures are explained below in the part of this document on IAC. For an overwrite import, use -1 to retain the object's previous pseudo signature. On a new import, you must provide a pseudo signature (which can be blank, indicating that the object is not linked).

(p) The <real signature> is a four character identifier, which is the real signature of the application to which the object is linked. The difference between real and pseudo signatures is explained below. For an overwrite import, use -1 to retain the object's previous real signature. On a new import, you must provide a real signature (which can be blank, indicating that the object is not linked).

(q) The <alias> is a short string containing the "canonical" name of the application to which the object is linked. For instance, if the object is linked to a program that happens to be called "ALOHA 5.0 testing version", the <alias> might be just ALOHA. Whenever you provide an <application path>, you must provide an <alias>. Otherwise you can leave the <alias> blank to retain the previous alias of the object.

(r) The <application path> is the full path name of the application to which the object is linked. A full path name of the form "disk name:folder name:subfolder name: ... :file name". This path name is self-correcting in the sense that if MARPLOT tries to use it and finds that it is invalid, the user is asked to locate the file and the path is repaired. Thus, you can export a linked object on one computer and import it on another computer. The path names will be incorrect but will be corrected upon their first use. You can leave the application path blank or use -1 to retain the object's previous application path.

(s) The <document path> is the full path name of the document to which the object is linked. It is self-correcting in the same way as the <application path>. You can leave the application path blank or use -1 to retain the object's previous application path.

(t) The <record> is an integer whose absolute value is smaller than 2 billion. It indicates the record within the specified document to which the object is linked. On an overwrite import, you can use -1 to retain the object's previous record. On a new import you must provide a value (use 0 if the object is not linked).

(u) The <note> is a sequence of up to 80 characters containing an arbitrary note about the object. The note must all be on one line and therefore cannot contain any return characters. For an overwrite import, use -1 to retain the object's previous note. On a new import, you must provide a note (which can of course be blank).

Example:

This sample import file contains two objects, each on a different overlay. They are both linked, although the object linked to ALOHA is not linked to a document. The second object is a polygon and thus its second line contains the polygon points.

```
1
* 1 Scenarios
* 0 Hospitals
A6CBE00FA0060404, 0, 5, 38.929328, 77.624480, 38.929328, 77.624480,14, 3, 9, 0, 0, 1, 1, 52, St. Mark's Hospital
   SPNT
   SPNT
   SuperPaint 2.0
   HD-120:Applications:SuperPaint:SuperPaint 2.0
   HD-120:Applications:SuperPaint:graph
   0
   This is the note for this symbol object.
A73158EFCC7044F0, 1, 6, 38.927548, 77.566264, 38.897768, 77.512048,1, 3, 9, 0, 4, 1, 1, 0, Facilities #CC7044F0
   5, 38.927548, 77.548000, 38.897768, 77.566264, 38.901768, 77.512048, 38.917768, 77.512616, 38.927548, 77.548000
   ALHA
   ALH5
   ALOHA
   HD-120:Newest CAMEO:ALOHA Folder:ALOHA

   0
   This plume was created by ALOHA.
```

Map file formats

Note: In short and long integers, the most significant byte is on the left (Mac style, opposite of PC style).

OBJ(object) files

1: length of object (not counting the four bytes for the length) (long)
owner code (long)
modifier code (long)
location code (state + county for TIGER objects) (5 bytes)
modification date (number of seconds since midnight January 1, 1904 = long)
version flags (nth even bit = app #n can read obj; nth odd bit = app #n can write obj) (long)
type (high bit = 0, reserved for future use) (byte)
id (8 bytes)
object 1 low-long + 180000000 (long) low long/ lat used to hold ID of original
object 1 low-lat + 90000000 (long) for alias objects
object 1 hi-long + 180000000 (long)
object 1 hi-lat + 90000000 (long)
digitization scale (long)
CFCC (long)
FIPS place-code (place code of majority of segments,
or 0 if no place;high order bit set if in more than one place) (long)
state-county (long) // holds the RGB values for RGB-colored objects (see MARPLOT colors)
alias count (byte)
length of prefix (total number of bytes allocated for prefix) (byte)
prefix (variable, might contain 0-terminator and thus not use all bytes allocated)
length of name (total number of bytes allocated for name) (byte)
name (variable, might contain 0-terminator and thus not use all bytes allocated)

2: POINT OBJECTS:
color (4 bits) + line-width (4 bits) (total = byte)
symbol (byte)

RECT/ CIRCLE OBJECTS:
color (4 bits) + line-width (4 bits) (total = byte)
line-pat (4 bits) + fill-pat (4 bits) (total = byte)

PICT OBJECTS:
frame (byte)
file name (32 bytes)

TEXT OBJECTS:
color (4 bits) + frame (4 bits) (total = byte)
font (short)
style (byte)
length of text (short)
text (variable, might end with 0-space)

POLYLINE/ POLYGON OBJECTS:
color (4 bits) + line-width (4 bits) (total = byte)
line-pat (4 bits) + fill-pat (4 bits) (total = byte)
number of segments (long)
segment 1 long + 180000000 (long; high order bit: 1 = FROM, 0 = TO)
segment 1 lat + 90000000 (long)
segment 1 flags (see below) (short)

segment 1 attribute 0 (long)
. . .
segment 1 attribute n (n < 15) (long)
< repeat for number of segments; possibility of extra 0-filled segment space at the end >

flags:	bit	indicates presence, in order, of
	0	TLID
	1	CFCC
	2	VERS (+ polyID * 100 for TIGER polygons)
	3 - 4	available for use by MARPLOT users
	5	start address left
	6	start address right
	7	end address left
	8	end address right
	9	zip code left
	10	zip code right
	11 - 14	reserved for future use
	15	segment is invisible (no corresponding attribute long is needed or used)

SUM (summary) files

offset (long) / / byte index of start of object in OBJfile (negative -> object is deleted)
low-long + 180000000 (long) <-
low-lat + 90000000 (long) <- all 0 for
hi-long + 180000000 (long) <- alias objects
hi-lat + 90000000 (long) <-
id (8 bytes)
first four characters of object's name (null terminated if < 4 chars) (4 bytes)
< repeat for number of objects >

SM2 (summary of summary) files

This file may not always exist for a given layer on a given map. When it does, it is a list of bounding rectangles, one rectangle for each 1000 summary records. Each rectangle is the union of its 1000 constituent object rectangles. The format of each rectangle is:

low-long + 180000000 (long)
low-lat + 90000000 (long)
hi-long + 180000000 (long)
hi-lat + 90000000 (long)

1000 * <number of SM2 records> may be less than <number of SUM records>, but not greater.

NNX (name index) files

NOTE: NNX files are no longer used in MARPLOT.

This file may not always exist for a given layer on a given map. When it does, it applies to all layers whose layer names match the prefix of the NNX file, possibly including an extension in parentheses. For instance, if there is an NNX file "ROADS.NNX", this file contains is an index into both the "ROADS" and "ROADS(MAJOR)" layers on that map. The NNX file is a list of pairs. Each pair contains a long, which is an offset into an OBJfile, and a small integer n, where n determines which of the layers with the same name (modulo parenthetical extensions) the object belongs to. n ranges from 1 to the number of layers with the given name. n indexes the layers in alphabetical order. For instance, in the case of the roads example, if n = 1 the object is on the "ROADS" layer, and if n = 2, the object is on the "ROADS(MAJOR)" layer, since it is alphabetically second.

Here is the format:

offset (long)
layer n (byte)
< repeat for number of objects in union of files with the given name >

The file is ordered according to the following sort expression:

name + prefix + place name + CFCC

LYR (layer information) files

layer name (32 chars)
layer's world rectangle on this map
low-long + 180000000 (long)
low-lat + 90000000 (long)
hi-long + 180000000 (long)
hi-lat + 90000000 (long)
number of object on layer on this map (long)

MAP (map information) files

map name (42 chars)
map ID (long; unused)
default location / owner (5 characters) ***
inUse (char; unused)
searchMe (char)
intersectMe (char)

*** either a state-county code or
SSSS0, where SSSS = owning application signature (first char non-digit)number of object on layer on this map (long)

FNT (alternate font) files

This optional file contains just a string of characters that specify the name of the font to be used on the given layer (on all maps, not just the map in which this file is included) in place of the MARPLOT font.

VEW (view information) files

view name (32 chars)
map map (42 chars)
view's world rectangle
    low-long + 180000000 (long)
    low-lat + 90000000 (long)
    hi-long + 180000000 (long)
    hi-lat + 90000000 (long)
scale at which view was saved (long)
user owned flag (char; unused)
owned flag (char; unused)
fileName (32 chars; unused)
volume reference number (short; unused)
directory ID (long; unused)

MSC (MARPLOT search collection) files

These files contain one line per object. Each line has the format:

"<map name>" "<layer name>" "<id>" <flags> <offset>

Where
<map name> is the name of the object's map
<layer name> is the name of the object's layer
<id> is the object's 16-digit MARPLOT id number
<flags> is an integer meaningful to MARPLOT
<offset> is the offset in the given OBJfile of the object

Note: <offset> is currently unused; for each object, MARPLOT searches for the given <id> on the given layer file, if it exists.

Note: <flags> is a combination of the following bits:

```
OP_SORTED = 1       ; used internally by MARPLOT
OP_ALIAS = 2        ; object is an ALIAS
OP_MARKED = 4       ; not used
```

VWR and MNU files

These are files that store information about friend applications and their Sharing menus.

PROG.VWR contains the "business card" information about a friend application.

It has the form:

```
signature
pseudo-signature
application name
application path
document path
```

For example, here is a file called CAMEO.VWR:

```
WILD
CAMO
CAMEO
C:\APPS\FOXPRO
C:\CAMEO\CAMEO.PRG
```

Not all applications need a corresponding document; in fact most don't. In this case the fourth line can just be blank (but you need to extra return).

PROG.MNU contains the text of the Sharing menu for the friend.

It has the form:

```
signature
pseudo-signature
menu name
menu item 1
...
menu item n
```

For example, here is a file called CAMEO.MNU:

```
WILD
CAMO
CAMEO
Help...
-
Get Info
Link
-
Go to CAMEO
```

If a line starts with a dash, it represents a dividing line in the menu.

CDMAPS files

These files are used by MARPLOT to access large collections of maps on CDs or other media without having to read each .MAP and .LYR file at startup.

The format is as follows: The first line gives the volume label (disc name) of the CD or drive containing the maps. This is followed by n lines, one for each layer that is represented in any of the maps on the CD. Each line has the following format:

```
N FILENAME, Layer Name
```

where `N` is a unique character, such as a digit, used to flag the layer throughout the file, `FILENAME` is the name of the files, without the suffix, that the layer is stored in on the disc, and `Layer Name` is the name of the layer as it appears in MARPLOT.

Following these layer definitions is one line that contains just the word `MAPS`, which flags the start of the maps section. For each map, the first line is of the format:

```
path FIPS Map Name
```

where `path` is the DOS-format path to the map's directory on the disc, starting from but not including the root directory, `FIPS` is the 5-digit FIPS state/ county code for the map, and `Map Name` is the name of the map as it appears in MARPLOT.

After this first line, there is one line for each layer represented on the map. Each of these layer lines is of the format:

```
N LOLONG LOLAT HILONG HILAT NUMOBJECTS
```

where is N is the unique character corresponding to the layer, as specified at the top of the file, the next 4 fields are the MARPLOT-format low longitude, low latitude, high longitude and high latitude of the bounding world rectangle of the layer on the map, NUMOBJECTS is the number of objects in the layer on the map. To convert to the MARLOT fomat from a decimal lat/ long value (using negative numbers in the western and southern hemispheres), write the value to six decimal place accuracy, but leave out the decimal, then add 180000000 to longitude values and 90000000 to latitude values.

A sample file follows.

```
Atlas
4 PLACES, Places
8 WATER, Water
MAPS
\SHIO\MAPS\09\09007\ 09007 MIDDLESEX COUNTY, CT
4 107246597 131177673 107693992 131646900 16
8 107250747 131177673 107692817 131644699 870
\SHIO\MAPS\09\09009\ 09009 NEW HAVEN COUNTY, CT
4 106672447 131087009 107471633 131644214 19
8 106672447 131087009 107471633 131643100 1012
.
.
.
```

MARPLOT ID numbers

MARPLOT ID numbers are 8 bytes. They are often interpreted as two sequential long values (the "hi" and "lo" fields of the ObjectID structure) or as a 16-character hex string.

A) ID numbers randomly assigned by MARPLOT.

When an object is created by hand in MARPLOT, or is imported with an ID of -1, MARPLOT assigns the objects a new ID number which is designed to be random enough to be universally unique. The following function is used to generate the ID.

```
void GenerateNewID(ObjectID *id)
{
        static long ticks = 0, count = 0;
        unsigned long seconds;

        if (!ticks) ticks = TickCount();
        GetDateTime(&seconds);
        id->lo = (ticks << 16) + (++count);
        id->hi = seconds;
}
```

This function builds the 8 random bytes out of (1) the computer's tick count the first time GenerateNewID() is called during this run of MARPLOT, (2) the computer's second count at the time the ID is being generated, and (3) a running count of the total number of objects made during this run of MARPLOT.

B) ID numbers pre-set to help an external application identify an object. (In the following, ssccc stands for the two-digit state code and three-digit county code.)

- A Census Block Group polygon object is output from the TIGER Translator with its
  id = "000Asscccttttxxb"
  where tttt is the four-digit basic Census Tract number (padded on the left with 0's)
  xx is the two-digit Census Tract suffix (padded on the left with 0's)
  b is the first digit of the Block Number of the blocks that make up the group
- A city/ place polygon object is output from the TIGER Translator with its
  id = "00000Bssccpppppp"
  where ppppp is the five-digit FIPS place code (padded on the left with 0's)
- A county polygon object is output from the TIGER Translator with its
  id = "0000000000Cssccc"
  NOTE: A thinned county polygon object has the same ID as its complete couterpart.
- A landmark polygon object, such as a water body or university, is output from the TIGER Translator with its
  id = "Dsscccpppppppppp"
  where pppppppppp is the ten-digit polygon code (padded on the left with 0's)
- A landmark point object, such as a school or lighthouse, is output from the TIGER Translator with its
  id = "Esscccllllllllll"
  where lllllllll is the ten-digit landmark code (padded on the left with 0's)
- A polyline object, such as a road, is output from the TIGER translator with its
  id = "Fsscccttttttttt"
  where ttttttttt is the ten-digit TIGER line ID of one of the segments making up the object; the segment chosen is the one with the lowest TIGER line ID number of all the segments making up the object (padded on the left with 0's)

- An alias object for a polyline object is output from the TIGER translator with its
  id = "FFn000ttttttttt"
  where n (1 - 9) is the alias count (e.g., 5 for the 5th alias of the original)
  ttttttttt matches the ttttttttt part of the of the original (padded left)

Vector field objects

If a polyline object has a CFCC code of "V00" it is treated specially by MARPLOT as a vector field. A vector field object is assumed to be a a sequence of FROM-TO point pairs. Vector fields are treated specially in the following ways:

• The name of the object is always in the format "`1 in = 2 m/s`". Here, the "`in`" may be replaced by "`cm`" if desired, the "`2`" can be any value, and the "`m/s`" may be replaced by any unit desired. In other ways, the name must match the given format exactly, including spaces.

• The "digitization scale" field of an object normally stores the number N to indicate that the object's original digitization resolution was 1:N. For a vector fild object, the digitization scale indicates the maximum length of a vector component unit in millionths of a degree latitude. For instance, if this number is 1000000, MARPLOT can assume it will never have to display this object at a scale such that a unit length of any vector's u or v component, when translated from inches on the screen to real world coordinates at the current scale, exceeds one degree latitude (60 nautical miles). If the user zooms out far enough that this condition is violated, MARPLOT displays a gray box in place of the vector field. The purpose of this scale limit is the enable MARPLOT to deal efficiently with vector objects by placing an upper bound on their real-world extents. It is up to the application that creates the vector field object's MIE file to set this number appropriately. Using a large number like 1000000 will work for most purposes. Using the smallest number possible will give maximum efficiency.

• The "state-county" field of an object normally stores a 5-digit number that encodes the state and county to which the object belongs (since MARPLOT data is often derived from county-based census data). For a vector field object, this number is a scale factor with an implied six decimal places. Thus, 1000000 indicates a scale factor of 1, meaning no scaling, while 2000000 indicates a scale factor of 2, meaning that each vector's u-v values are considered to be twice their original value.

• When a vector field object is created during an MIE import (this is the only way to create a vector object), MARPLOT checks to make sure the name is in the proper format, and sets the state-county/ scale factor to 1000000. It uses the imported digitization scale/ max length value to store the vector field using specially encoded latitude/ longitude coordinates. The MIE record is assumed to contain a list of FROM-TO point pairs, with the FROM points in latitude/ longitude coordinates, and the TO points in the vector field's u-v units.

• When a vector field field is drawn, the length of an individual vector is based on that vector's u-v values, and the scale specified by the object's name. If the user renames the vector to change this scale, the field is redisplayed accordingly (MARPLOT will not allow the user to enter an vector field name that is not in the proper format). If the user clicks and drags the end of any vector in the field, the state-county/ scale factor is modified accordingly. Conceptually, this can be thought of as simultaneously scaling the u-v values of all vectors in the field.

• When a vector field object is exported back into an MIE file, any scaling that has taken place is reflected in scaled u-v vlaues, and the scale written to the MIE file is always 1000000.

MARPLOT colors

Millions of colors

With the release of MARPLOT 3.2, MARPLOT allows each object to carrys its own RGB color value, allowing for millions of colors. You can assign colors to objects by hand, using the Color menu in MARPLOT's menu bar and the color popup menu in the Object Settings dialog box, or via import. The three MARPLOT import formats (MIE, Simple Point, and GENERATE) have been extended to allow an RGB value in the fields where previously a short (1-16) integer value (or an equivalent constant name) was allowed. In place of a constant such as `RED`, for example, you could use the string of characters `R200G100B50`. This value represents the RGB color with a red value of 200, a green value of 100, and a green value of 50. These values are each in the range of 0 (least bright) to 255 (most bright).

NOTE: Internally, MARPLOT stores the RGB values for an object in its state-county field. The high byte of this long is used for flags. If bit 0 of the high byte is set, the field represents an RGB value (as opposed to an obsolete state-county value). If bit 1 of the high byte is set, the RGB value is active and overrides the object's old style (1 – 16) color. The remaining three bytes hold, from right to left, the red, green and blue values. Note that it is possible to transfer RGB-colored objects in import files without the use of the extended color format (e.g., `R200G100B50`) in the color field; the state-county field contains all of the information necessary and the extended color format is provided only for encoding convenience.

"Ideal" and ESI colors

MIE value	MARPLOT 3.0 "ideal" color	MARPLOT 3.2 color	Old ESI color	New ESI color (* = change)
1	black	black	black	black
2	white	white	white	white
3	dark-gray	dark-gray	brown	brown
4	gray	gray	purple	purple
5	light-gray	light-gray	light-purple	light-purple
6	brown	brown	blue	blue
7	light-brown	light-brown	light-blue	light-blue
8	dark-green	dark-green	blue-green	blue-green
9	green	green	green	green
10	light-blue	light-blue	green-yellow	green-yellow
11	blue	blue	yellow	yellow
12	purple	dark-blue (MIE = 15)	orange	light-brown *
13	pink	purple (MIE = 12)	red	orange *
14	red	pink (MIE = 13)	pink	red *
15	orange	red (MIE = 14)	light-brown	pink *
16	yellow	yellow	off-white	TBA

Polygon union

This section describes some technical problems related to polygon union, and how these problems are solved by MARPLOT 3.0 and MARPLOT 3.2.

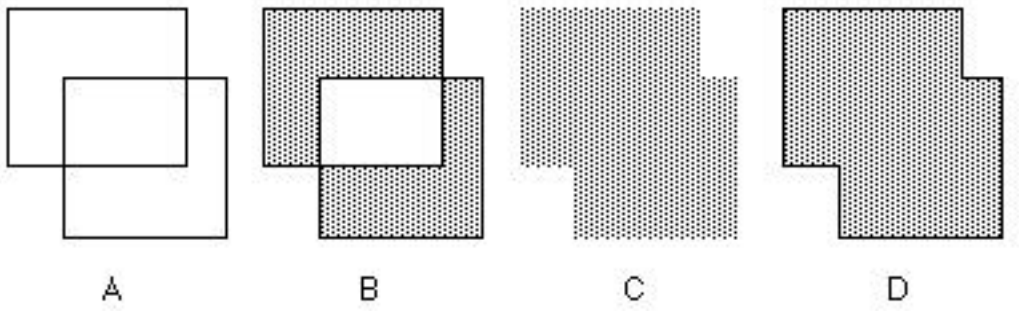


Figure A above shows two overlapping polygons. When the union of these two polygons is computed, we would like to get the polygon in figure D. In most cases, it is not difficult to compute results as in figure D. However, in the case of very convoluded polygons, and especially polygons that have one or more sides in common, computing the "right" answer as in figure D is quite difficult.

A backup method is simply to take the two component polygons and "throw them together" in a "poly-polygon," that is, a single polygon object that retains both component polygons as pieces. This would work, except that normally, when a polygon in MARPLOT is made up of more than one piece (more than one connected "island", "loop" or "chain"), it is the case for any two given pieces either that they do not overlap at all (imagine a lake that is made up of two disjoint water bodies), or that one is included entirely in the other (imagine a lake with an island in the middle). MARPLOT's general rule is that when two pieces of a poly-polygon overlap, the overlapping area is treated a hole (again, think of a lake with an island). But in the case of unions, thinking of the overlapping area as a whole is not what we want (see figure B above).

The solution to this problem in MARPLOT is to give these union polygons a line pattern of Ø (null or white). This is a flag for MARPLOT not to treat the overlapping areas as holes. Thus, if we give the polygon in figure B a Ø line pattern, it appears in MARPLOT as in figure C. Figure C looks good, but remember that there are really two separate pieces there. This fact can be ignored until we try to compute the polygon's area. MARPLOT 3.0 will report an area that is too large, being the sum of the areas of the two components. MARPLOT 3.2 simply does not report an area at all for polygons that have a Ø line pattern.

With all of this as background, here is the situation with polygon unions in MARPLOT 3.0 and MARPLOT 3.2.

Because of the possibility of being unable to compute the "right" answer (as in figure D), MARPLOT 3.0 always makes a poly-polygon with a Ø line pattern (figure C). These polygons look OK, but their area is reported incorrectly by MARPLOT 3.0 and not reported at all by MARPLOT 3.2.

When MARPLOT 3.2 computes a union, it checks whether the two pieces have segments in common. If so, it gives up on computing the "right" answer, and creates a poly-polygon as in MARPLOT 3.0. If there are no overlapping segments, it creates (or at least attempts to create) the right answer as in figure D.

When MARPLOT (version 3.0 or version 3.2) computes an "envelope" or "buffer zone" polygon around a polyline, this is really just a complicated case of several successive polygon unions. Because these unions almost always involve shared or very close segments, both versions of MARPLOT revert to the poly-polygon/ Ø line pattern solution. This means that the areas of these envelope polygons is reported incorrectly by MARPLOT 3.0, and not at all by MARPLOT 3.2.

More on polygon areas

When MARPLOT 3.0 computes the area of a poly-polygon, it simply adds up the areas of the component pieces. This gives the right answer if the components pieces are all disjoint, but gives the wrong answer if the component pieces contain one another (i.e., if there are holes in the polygon).

MARPLOT 3.2 checks each component piece to see how many other component pieces it is contained within. If it is contained within an odd number of the other component pieces, the given piece is considered to be a hole, and its area is subtracted from the total area of the polygon. Otherwise the area of the given piece is added to the total area of the polygon. This when pieces completely contain one another, but there are still problems when two pieces intersect but one is not completely contained within the other. In this latter case MARPLOT 3.2 will give an incorrect area (but remember that if the polygon's line pattern is Ø MARPLOT 3.2 does not report an area at all).

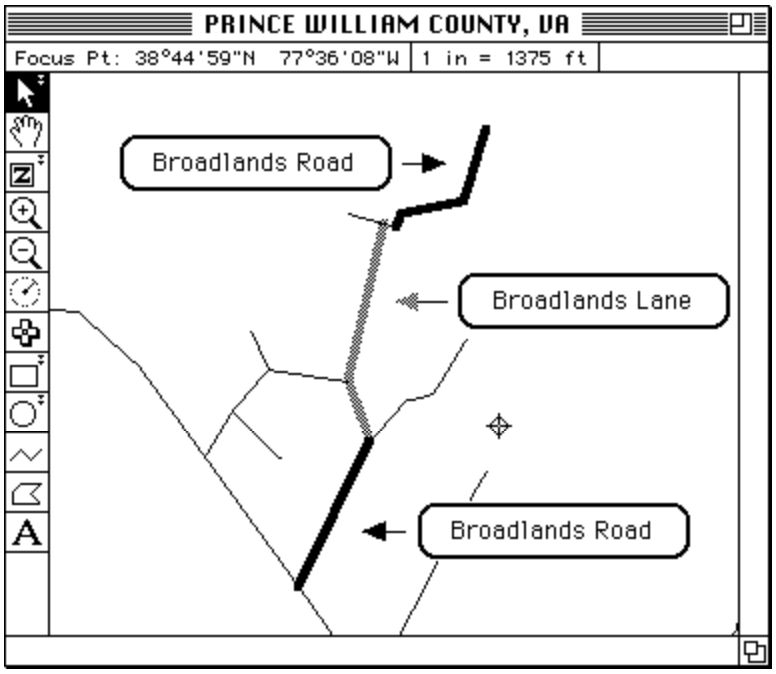
Advanced example: Editing roads using Export/Import

MARPLOT provides a number to tools for editing roads: dragging points, inserting points, moving points to the marked point, setting segment attributes.

However, there are a number of changes you might want to make, to roads or to other types of objects, that cannot be done, or cannot be done in an efficient manner, using MARPLOT's built-in editing mechanisms. In this section, we present one such example, and explain how you can effect the desired change using MARPLOT Export and Import functions.

WARNING: Back up any map directory (folder) that you plan to alter using Export and Import. It is very easy to lose data by using these functions incorrectly.

The sample Prince William County map includes two roads, Broadlands Road and Broadlands Lane, which are shown in the picture below. The objects have been modified using MARPLOT's Object Settings dialog box to make them stand out. The gray piece in the center is Broadlands Lane. Broadlands Road is split. Part of it is north of the Lane, and part of it is south of the Lane.



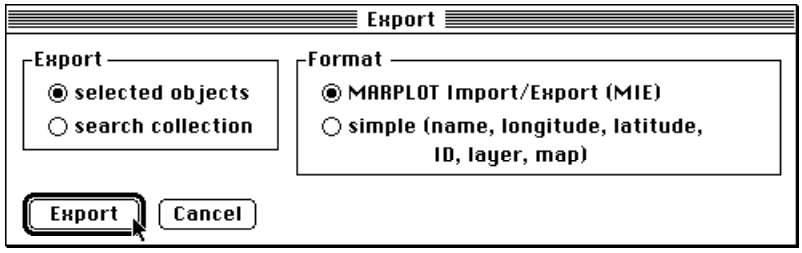
Suppose that the name "Broadlands Lane" is outdated or incorrect. Suppose that we want to consolidate all of the thick segments in the picture into a single road object called Broadlands Road. It is possible to do this using the tools seen in the previous example, by extending one or both halves of Broadlands Road, segment by segment, setting the address ranges to match the corresponding segments of Broadlands Lane, and finally deleting Broadlands Lane. In this particular case, Broadlands Lane happens to have only two segments, so this approach could be done without too much pain. However, if there were many segments in Broadlands Lane, this segment-by-segment editing would take forever.

A smarter and more general approach is to use Export and Import. The idea is to select one or more objects, export them to a MARPLOT Import/ Export (MIE) text file, use a text editor to make changes

to the MIE file, and then import the modified MIE file back into MARPLOT, which will cause the old objects to be replaced by the new objects in the file. This technique is very powerful, since any type of object editing, in princple, can be accomplished this way.

For the purpose of this example, very little understanding of the MIE format is required.

To start, we select both Broadlands Road and Broadlands Lane on the map by clicking each while holding down the shift key. Then we choose the Export command from the File menu, and choose to write the objects to an MIE file called TWOROADS.MIE.



Then, using a text editor (such as Notepad on Windows or Simple/ Teach Text on Macintosh), we open TWOROADS.MIE. Here is how the text of the file appears in the text editor.

```
"CENS" "USER" "51153" "04/19/95" 2
"" "Broadlands Lane" 0 "Roads" "PRINCE WILLIAM COUNTY, VA" POLYLINE
"F511530076503782" 100000 "A40" 0 ONLY 51153
BLACK 3 DARKGRAY WHITE { { FROM -77.605904 38.754900 } { TO -77.607104
38.750800 { TLID 76503794 } { CFCC "" } { VERS 5 } { SAL 9485 } { SAR
9506 } { EAL 9277 } { EAR 9174 } { ZCL 22123 } { ZCR 22123 } } { TO
-77.606400 38.749200 { TLID 76503782 } { SAL 9477 } { SAR 9476 } { EAL
9499 } { EAR 9498 } } }

"CENS" "USER" "51153" "04/19/95" 2
"" "Broadlands Road" 0 "Roads" "PRINCE WILLIAM COUNTY, VA" POLYLINE
"F511530076503783" 100000 "A40" 0 ONLY 51153
BLACK 3 BLACK WHITE { { FROM -77.606400 38.749200 } { TO -77.608800
38.745400 { TLID 76503783 } { CFCC "" } { VERS 5 } { SAL 9501 } { SAR
9500 } { EAL 9699 } { EAR 9698 } { ZCL 22123 } { ZCR 22123 } } { FROM
-77.602504 38.757400 } { TO -77.603200 38.755500 { TLID 76503796 } { SAL
9101 } { SAR 9100 } { EAL 9299 } { EAR 9298 } } { TO -77.605304
38.755200 } { TO -77.605504 38.754800 } }
```

We can see that there are two objects in the file. The first "paragraph" is the information for Broadlands Lane. The second paragraph is Broadlands Road. We see that Broadlands Land is on the "Roads" layer of the "Prince William County, VA" map, that it's a POLYLINE, and that its pattern is DARKGRAY.

The information inside the { braces } specifies the latitude/ longitude values for the points of the roads, along with the segment attributes such as addresses and ZIP codes. We can select this coordinate/ segment data for Broadlands Lane in the text editor.

```
"CENS" "USER" "51153" "04/19/95" 2
"" "Broadlands Lane" 0 "Roads" "PRINCE WILLIAM COUNTY, VA" POLYLINE
"F511530076503782" 100000 "A40" 0 ONLY 51153
BLACK 3 DARKGRAY WHITE { { FROM -77.605904 38.754900 } { TO -77.607104
38.750800 { TLID 76503794 } { CFCC "" } { VERS 5 } { SAL 9485 } { SAR
9506 } { EAL 9277 } { EAR 9174 } { ZCL 22123 } { ZCR 22123 } } { TO
-77.606400 38.749200 { TLID 76503782 } { SAL 9477 } { SAR 9476 } { EAL
9499 } { EAR 9498 } } }

"CENS" "USER" "51153" "04/19/95" 2
"" "Broadlands Road" 0 "Roads" "PRINCE WILLIAM COUNTY, VA" POLYLINE
"F511530076503783" 100000 "A40" 0 ONLY 51153
BLACK 3 BLACK WHITE { { FROM -77.606400 38.749200 } { TO -77.608800
38.745400 { TLID 76503783 } { CFCC "" } { VERS 5 } { SAL 9501 } { SAR
9500 } { EAL 9699 } { EAR 9698 } { ZCL 22123 } { ZCR 22123 } } { FROM
-77.602504 38.757400 } { TO -77.603200 38.755500 { TLID 76503796 } { SAL
9101 } { SAR 9100 } { EAL 9299 } { EAR 9298 } } { TO -77.605304
38.755200 } { TO -77.605504 38.754800 } }
```

Notice that we select everything except the starting and ending braces.

Using the cut and paste functions in the editor, we move the selected text from Broadlands Lane to Broadlands Road.

```
"CENS" "USER" "51153" "04/19/95" 2
"" "Broadlands Lane" 0 "Roads" "PRINCE WILLIAM COUNTY, VA" POLYLINE
"F511530076503782" 100000 "A40" 0 ONLY 51153
BLACK 3 DARKGRAY WHITE {  }

"CENS" "USER" "51153" "04/19/95" 2
"" "Broadlands Road" 0 "Roads" "PRINCE WILLIAM COUNTY, VA" POLYLINE
"F511530076503783" 100000 "A40" 0 ONLY 51153
BLACK 3 BLACK WHITE { { FROM -77.605904 38.754900 } { TO -77.607104
38.750800 { TLID 76503794 } { CFCC "" } { VERS 5 } { SAL 9485 } { SAR
9506 } { EAL 9277 } { EAR 9174 } { ZCL 22123 } { ZCR 22123 } } { TO
-77.606400 38.749200 { TLID 76503782 } { SAL 9477 } { SAR 9476 } { EAL
9499 } { EAR 9498 } } { FROM -77.606400 38.749200 } { TO -77.608800
38.745400 { TLID 76503783 } { CFCC "" } { VERS 5 } { SAL 9501 } { SAR
9500 } { EAL 9699 } { EAR 9698 } { ZCL 22123 } { ZCR 22123 } } { FROM
-77.602504 38.757400 } { TO -77.603200 38.755500 { TLID 76503796 } { SAL
9101 } { SAR 9100 } { EAL 9299 } { EAR 9298 } } { TO -77.605304
38.755200 } { TO -77.605504 38.754800 } }
```

Notice that we inserted the text just after the first brace in Broadlands Road.

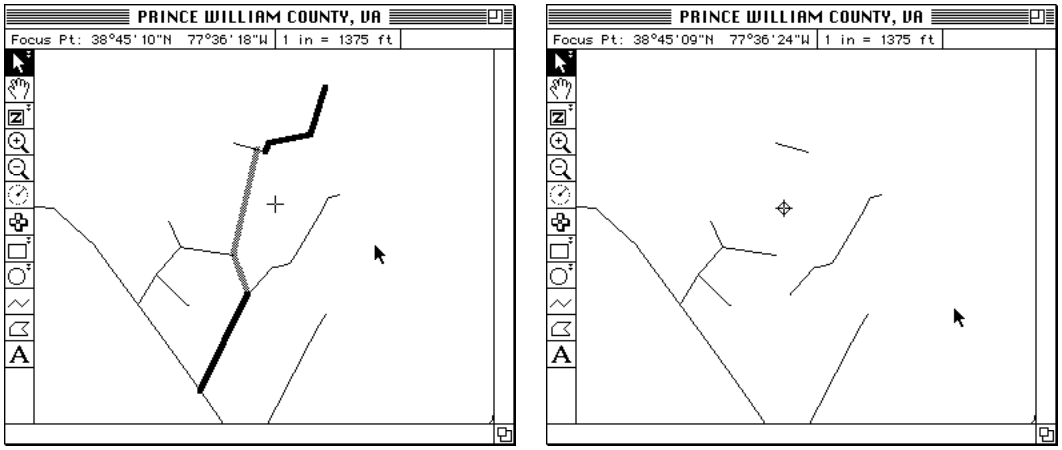
We then delete the remaining text of Broadlands Lane.

```
"CENS" "USER" "51153" "04/19/95" 2
"" "Broadlands Road" 0 "Roads" "PRINCE WILLIAM COUNTY, VA" POLYLINE
"F511530076503783" 100000 "A40" 0 ONLY 51153
BLACK 3 BLACK WHITE { { FROM -77.605904 38.754900 } { TO -77.607104
38.750800 { TLID 76503794 } { CFCC "" } { VERS 5 } { SAL 9485 } { SAR
9506 } { EAL 9277 } { EAR 9174 } { ZCL 22123 } { ZCR 22123 } } { TO
-77.606400 38.749200 { TLID 76503782 } { SAL 9477 } { SAR 9476 } { EAL
9499 } { EAR 9498 } } { FROM -77.606400 38.749200 } { TO -77.608800
38.745400 { TLID 76503783 } { CFCC "" } { VERS 5 } { SAL 9501 } { SAR
9500 } { EAL 9699 } { EAR 9698 } { ZCL 22123 } { ZCR 22123 } } { FROM
-77.602504 38.757400 } { TO -77.603200 38.755500 { TLID 76503796 } { SAL
9101 } { SAR 9100 } { EAL 9299 } { EAR 9298 } } { TO -77.605304
38.755200 } { TO -77.605504 38.754800 } }
```

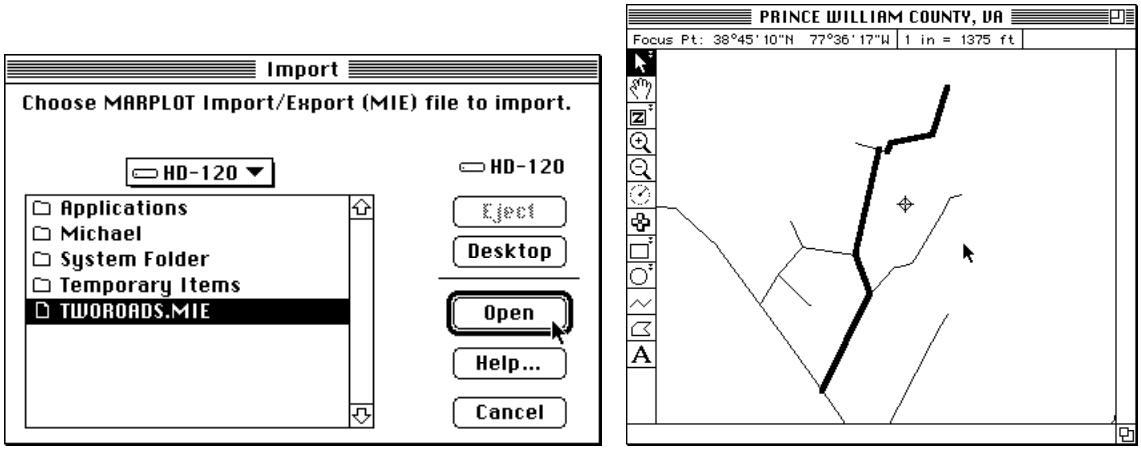
That's it. The segments of Broadland Lane have been assimilated into Broadlands Road.

We now save the modified TWOROADS.MIE file (be sure to save it in text-only format) and exit from the text editor.

Back in MARPLOT, we unlock the Roads layer, and then delete both Broadlands Lane and Broadlands Road (in fact, it is not necessary to delete Broadlands Road, because the Import would replace the old one with the new one, but it doesn't hurt to delete it now anyway; we do have a backup of this map, right?).



Then we choose Import from the File menu and select the modified TWOROADS.MIE file.



The Import occurs and we see the new Broadlands Road. If we had made an error in editing the text of the MIE file, MARPLOT would give an error message during the Import, and we would have to go back to the text editor to fix the file before trying to import it again.

The point of this example has been to show that using Export and Import often provides a convenient and powerful way to perform MARPLOT edits that would otherwise be very difficult. For some of these edits, only superficial understanding of the MIE format is required. For others, a more detailed understanding is needed.

Interapplication communication (IAC) dictionary

Introduction

This is the MARLOT IAC dictionary. It specifies the messages sent to and from MARPLOT via Apple Events on the Macintosh and via the NOAA IAC DLLs on Windows.

Overview of CAMEO/MARPLOT/ALOHA IAC Mechanism

These programs send each other messages. On the Macintosh this is done through Apple Events. On Windows it is done through a combined mechanism of DLL calls, window messages and file passing. In both cases, an IAC message can be thought of much like a function call to another program. The name of the function and each of its parameters are specified by 4–character strings. Unlike a regular function call, some of the parameters in an IAC message are sometimes optional. Also unlike a regular function call, all parameters in an IAC message are strings.

The exact mechanism for sending a message is technical and machine/ application-dependent. There are different low-level mechanisms for sending and recieving messages in Macintosh C programs, Macintosh Hypercard programs, Macintosh FoxPro programs, Windows 16-bit C programs, Windows 32-bit C programs and Windows FoxPro programs. Fortunately, the actual low-level code for sending and receiving the messages is short.

Besides sending each other messages, the programs need to be able to do things like launch each other, bring each other forward, check if another program is running, ask the user to locate a program that it knows about but can't find, etc. All of this is machine and application-dependent.

On both the Macintosh and Windows, there is the option for the client to wait for a reply to a message before continuing on. For a number of reasons, we don't use this option. When any CAMEO program sends a message, it simply sends it and goes on. If the application sends a response in another message a moment later, great, if not, oh well. The main drawback to this scheme is that applications can lose track of conversations that require a number of messages back and forth. However, this can be solved by the use of global status variables, or better yet by the use of the XTRA standard parameter.

Standard Parameters

Each message includes the four parameters MSSG, SIGN, PSIG and XTRA.

The data for the MSSG parameter is the 4-character "function name" for the IAC function that is being invoked.

The data for the SIGN parameter is the "signature" of the client (calling) application. This is Macintosh terminology for a 4-character code that identifies the application. MARPLOT's signature is MRP1, CAMEO's signature is WILD, and ALOHA's signature is ALH5. On Windows, the signatures is used as the server name of the application.

To allow for greater flexibility, and to accomodate some quirks of Hypercard, we use a two-level application identification scheme. Each application, in addition to its signature, also has a "pseudo-signature," which is sent in the PSIG parameter. MARPLOT's pseudo-signature is PLOT, CAMEO's pseudo-signature is CAMO, and ALOHA's pseudo-signature is ALHA.

Finally, each message includes a parameter called XTRA. When an application receives a message, it should store the contents of the XTRA parameter in a static/ global variable. When it sends a message, it should pass the current contents of the XTRA global again as the XTRA parameter. This allows the XTRA parameter to serve as a transaction identifier. An application can send off a request for information, tagged in a certain way through the use of the XTRA parameter. When it gets a reply from the other application, it can "remember" what it was doing, by looking at the XTRA parameter. Of course, in the case that an application sets the XTRA parameter, it cannot simultaneously return another application's previous XTRA parameter. Fortunately, we never want to do this, since an application is generally either "asking" or "responding," not both.

Mechanism on Macintosh (System 7 or higher)

To send a message:

Check if the receiving application is running. If not, try to launch it. Create an Apple Event of type 'NOAA', class 'AEVT'. The individual parameters are packed as Apple Event parameters, where the keyword of the parameter is the 4-character parameter name, and the data of the parameter is the data string for the IAC parameter. The parameters are added with AEPutParamPtr(), and the entire message is sent with AESend().

To recieve a message:

At startup, install an AE handler to handle 'NOAA', 'AEVT' events:

```
AEInstallEventHandler('NOAA', 'AEVT', MyHandler, 0, false);
```

In the event loop, process Apple Events:

```
case kHighLevelEvent: AEProcessAppleEvent(&event); break;
```

In MyHandler, extract parameters "MSSG", "SIGN", and "XTRA". Extract additional parameters as needed for the given message. Process message.

To see if an application is running:

```
Boolean SigToProcessInfo(char *sig, ProcessInfoRec *pInfo,
                            char *name, FSSpec *spec)
{
       char dummyName[32];
       ProcessSerialNumber PSN;
       FSSpec dummySpec;

       pInfo->processInfoLength = sizeof(ProcessInfoRec);
       pInfo->processName = (name ? name : dummyName);
       pInfo->processAppSpec = (spec ? spec : &dummySpec);

       PSN.highLongOfPSN = 0;
       PSN.lowLongOfPSN = kNoProcess;

       while (GetNextProcess(&PSN) != procNotFound) {
          GetProcessInformation(&PSN, pInfo);
          if (!strncmp((char *)&pInfo->processSignature, sig, 4))
              return TRUE;
       }

       return FALSE;
}

Boolean AppIsRunning(char *sig)
{
       ProcessInfoRec pInfo;

       return SigToProcessInfo(sig, &pInfo, 0, 0);
}
```

Mechanism on Windows

The message sent is a string of the following form:

param 1 name • param 1 • param 2 name • param 2 • . . . • param n name • param n ø

where • is the vertical tab character (ascii 11) ø is a null-terminator

param 1 is always MSSG
param 2 is always SIGN
param 3 is always PSIG
param 4 is always XTRA
of the remaining paramters, the largest should come last, for efficiency

The following constants are used below:

```
#define WM_IAC (WM_USER + 1)
#define NE_GET_ALL_MESSAGES 1
#define NE_APP_IS_RUNNING 2
#define NE_TRANSFER_MESSAGE 3
```

Mechanism used by MARPLOT (a 32-bit application)

Note: Currently MARPLOT is the only 32-bit application among the communicating NOAA Windows programs. In the future, when there is more than one 32-bit application, there will be a NOAA32 DLL analogous to the NOAA16 DLL. 32-bit applications will send messages via the NOAA32 DLL, which will communicate with the NOAA16 DLL via files. Eventually, when there are no 16-bit applications left, all applications will communicate via NOAA32, without file passing.

To start:

You may either assume that the NOAA_32.DLL is in the system directory, or, if your application has a copy of NOAA_32.DLL , it can check whether its copy is newer than the one currently in the system directory, and replace it if so.

Load the NOAA32 DLL and register with it by calling its NERegister() function. For example, here is how an application named MARPLOT, with signature `MRP1`, would register:

```
HINSTANCE gNOAADLLInst = NULL;

short CallNERegister(void) // 32 bit version
{
	Boolean doCopy = FALSE;
	short err;
	char theirPath[256], myPath[256];
	unsigned long theirTime, myTime;
	FARPROC proc;

	GetSystemDirectory(theirPath, 255);
	strcat(theirPath, "\\NOAA_32.DLL");
	_getcwd(myPath, 255);
	strcat(myPath, "\\NOAA_32.DLL");
	if (err = GetFileModDate(0, 0, theirPath, (long *)&theirTime))
		doCopy = TRUE;
	else
		if (!GetFileModDate(0, 0, myPath, (long *)&myTime))
			if (myTime > theirTime)
				doCopy = TRUE;

	if (doCopy)
		CopyFile(myPath, theirPath, FALSE);

	gNOAADLLInst = LoadLibrary(theirPath);

	if (!gNOAADLLInst) { SysBeep(1); return -1; }

	proc = GetProcAddress(gNOAADLLInst, "NERegister");
	if (proc) {
		if (err = proc((Ptr)"MRP1", // signature
				(HWND)mapWindow, // main window
				(Ptr)"MARPLOT",  // main window class  name
				(Ptr)0,             // reserved for future use
				(Ptr)"MARPLOT",  // application name
				(Ptr)0,             // reserved for future use
				(Ptr)0,             // reserved for future use
				(UINT)WM_IAC,       // or use your own value
				(WORD)0,            // wParam for WM_IAC
				(LONG)0))    // lParam for WM_IAC
			{ SysBeep(1); return err; }
	}
	else
		{ SysBeep(1); return -1; }

	return 0;
}
```

To send a message:

First make sure the receiving application is running. If not, launch it. Then:

```
short CallNESendMessage(Ptr toSignature, Ptr messageStr) // 32 bit
{
       short err;
       FARPROC proc;

       proc = GetProcAddress(gNOAADLLInst, "NESendMessage");
       if (proc)
              err = proc((Ptr)toSig,            // signature
                           (Ptr)messageStr,      // message
                           (UINT)0,        // reserved for future use
                           (Ptr)"MARPLOT", // application name
                           (Ptr)0);        // reserved for future use
       else
              err = -1;

       return err;
 }
```

To receive a message:

In your WndProc(), check for `WM_IAC` messages (whatever message you sent as the eighth parameter to `NERegister()`). When this message is received:

```
short HandleNEMessage(short wParam, long lParam) // 32 bit version
{
	Boolean result;
	long length;
	CHARH h;
	FARPROC proc;

	proc = GetProcAddress(gNOAADLLInst, "NEGetNextMessageLength");
	if (!proc) { SysBeep(1); return -1; }

	length = (long)proc((Ptr)"MRP1");

	if (!length) return 0;

	h = (CHARH)NewHandle(length + 1);
	if (!h)
		{ SysBeep(1); return -1; }

	proc = GetProcAddress(gNOAADLLInst, "NEGetNextMessage");
	if (!proc)
		{ SysBeep(1); DisposeHandle((Handle)h); return -1; }

	INDEXH(h, length) = 0;

	result = (Boolean)proc((Ptr)"MRP1",             // signature
				(Ptr)*h,        // data pointer
				(long)length);  // max length

	if (!result)
		{ DisposeHandle((Handle)h); return -1; };

	// process the message in h (you may want to queue h and
	// process it when you have free time)

	return 0;
}
```

To see if an application is running:

```
Boolean CallNEAppIsRunning(char *sig) // 32 bit version
{
	FARPROC proc;

	proc = GetProcAddress(gNOAADLLInst, "NEAppIsRunning");
	if (proc)
		return (Boolean)proc((Ptr)sig);

	SysBeep(1);

	return FALSE;
}
```

To quit:

```
void CallNEBye() // 32 bit version
{
	FARPROC proc;

	proc = GetProcAddress(gNOAADLLInst, "NEBye");
	if (proc)
		proc((Ptr)"MRP1",		// signature
			(HWND)mapWindow,	// main window
			(Ptr)class,		// main window class name
			(Ptr)0);	// reserved for future use
	else
		SysBeep(1);

	// freeing the DLL can cause complications and is unnecessary
	// FreeLibrary(gNOAADLLInst);
}
```

Mechanism used by 16-bit applications that communicate via the NOAA16 DLL

To start:

The file NOAA_16.DLL must be in the path. Load the NOAA16 DLL and register with it by calling its NERegister() function. For example, here is how an application named ALOHA, with signature ALH5, would register:

```
HINSTANCE gNOAADLLInst = NULL;

void CallNERegister(void) // 16 bit version
{
       char fullPath[256];
       FARPROC proc=NULL;

       if (gNOAADLLInst == NULL)
              gNOAADLLInst = LoadLibrary("NOAA_16.DLL");

       if ((UINT)gNOAADLLInst > 32) // we have the library
              if (proc = GetProcAddress(gNoaaDllInst, "NERegister")) {
                     GetPathToMe(fullPath); // path (incl. EXE file)
                     (*proc)((Ptr)"ALH5 ", // signature
                                   (HWND)myMainWnd, // window handle
                                   (Ptr)myMainClassName, // window class
                                   (Ptr)NULL, // reserved for future use
                                   (Ptr)"ALOHA", // "human readable" app name
                                   (Ptr)"ALOHA.exe", // EXE file name
                                   (Ptr)fullPath,
                                   // the last three parameters specify the
                                   // window message (including wParam and
                                   // lParam) that you want the DLL to send
                                   // to your application when it has an IAC
                                   // message for it
                                   (UINT)WM_MY_WAKEUP,
                                   (WORD)0,
                                   (LONG)0);
              }

}
```

To send a message:

First make sure the receiving application is running. If not, launch it. Then:

```
Boolean CallNESendMessage(Ptr toSignature, Ptr messageStr) // 16 bit
{
       FARPROC proc;
       long err = -1;

       if ((UINT)gNOAADLLInst > 32) { // we have the library
              proc = GetProcAddress(gNOAADLLInst,"NESendMessage");

       if (proc)
              err = (*proc)(toSignature, // signature of receiver
                            messageStr, // the message string
                            FALSE, // reserved for future use
                            NULL, // reserved for future use
                            NULL); // reserved for future use

       return err == 0;
 }
```

To receive a message:

In your WndProc(), check for `WM_MY_WAKEUP` messages (whatever message you sent as the eighth parameter to `NERegister()`). When this message is received:

```
Boolean HandleNEMessage(void) // 16 bit version
{
	FARPROC proc;
	Ptr message;
	long len, maxLength;

	if (gNOAADLLInst == NULL)
		gNOAADLLInst = LoadLibrary(gDllFileName);

	if ((UINT)gNOAADLLInst <= 32) // we do not have the library
		return FALSE;

	if (!(proc = GetProcAddress(gNOAADLLInst,
					"NEGetNextMessageLength")))
		return FALSE;

	if ((len = (long)(*proc)((Ptr)"ALH5")) <= 0) return FALSE;

	if (!(proc = GetProcAddress(gNOAADLLInst,
					"NEGetNextMessage")))
		return FALSE;

	if (!(message = MyAllocatePointer(len + 1))) return FALSE;

	if ((*proc)((Ptr)"ALH5", message, len + 1) == FALSE)
		{ MyFreePointer(messageString); return FALSE; }

	MyHandleIACMessage(&message);
	MyFreePointer(messageString);

	return TRUE;
}
```

To see if an application is running:

```
Boolean CallNEAppIsRunning(Ptr appSignature) // 16 bit version
{
	FARPROC proc;

	if (gNOAADLLInst == NULL)
		gNOAADLLInst = LoadLibrary(gDllFileName);

	if ((UINT)gNOAADLLInst > 32) { // we have the library
		proc = GetProcAddress(gNoaaDllInst,"NEAppIsRunning");
		if (proc)
			return (Boolean)(*proc)((Ptr)appSignature);
	}

	return FALSE;
}
```

To quit:

Send a message to the DLL to let it know you're quitting. Then free the DLL.

```
void CallNEBye(void) // 16 bit version
{
	char sigStr[6];
	FARPROC proc;

	if ((UINT)gNOAADLLInst > 32) { // we have the library
		proc = GetProcAddress(gNOAADLLInst, "NEBye");
		if (proc)
			(*proc)((Ptr)"ALH5", // signature
					(HWND)myMainWnd, // window handle
					(Ptr)myMainClassName, // window class
					(Ptr)NULL); // reserved for future use
		FreeLibrary(gNOAADLLInst);
		gNOAADLLInst= NULL;
	}
}
```

History of MARPLOT's Linking Scheme

In the beginning, there was MARPLOT DOS, which allows objects on the map to be linked to records in the CAMEO database. This was straightforward, since MARPLOT and CAMEO shared database files directly. MARPLOT used database files just like CAMEO, and knew all about CAMEO's database files, where they were, and which direct calls/ pass files to use to get CAMEO do display certain records. Similarly, there was an agreed-upon way for CAMEO to say "display this object" to MARPLOT.

With MARPLOT 1.0.1 on the Macintosh, we wanted to get away from the "hard-coded" connection between MARPLOT and CAMEO. The idea was that CAMEO was "just another database" that could contact MARPLOT and share information, but without MARPLOT knowing anything about CAMEO's database structure etc. To accomplish this, we developed a system of two-way links between applications that introduce themselves to each other. Each application was responsible for maintaining a complete reference to the linked record in the other application. This meant that CAMEO needed to know, for a given record, that it was linked to the object with identification number X on map M (in previous versions of MARPLOT, you could only work with one map at a time, so it was necessary to "open" the proper map in order to view an object it contained; also in previous versions, you could only link to user-created (as opposed to "basemap") objects, and since there were relatively few of these, it wasn't necessary for the database application to know about which layer the linked object was on). And MARPLOT needed to know, for a given object, that it was linked to record number R of document D of application A. Besides being burdensome for both programs, this double-linking scheme had a number of drawbacks. The most important was the possibility it presented of inconsistent links, where one application thought a record was linked to a certain object, and MARPLOT thought the object wasn't linked or was linked to a different record. Another problem was that it didn't allow the possibility of linking two records to a given object. For instance, two different database applications might have information on a given facility plotted on a map.

In MARPLOT (SPEARS), which is a special version of MARPLOT developed for the Coast Guard, we introduced a utilty program called Linker whose eventual role was to serve as a central link clearing house for all CAMEO-related applications. Linker would be launched in the background whenever any CAMEO application was running. The individual applications would not store the links themselves, but would ask Linker to record a link between, say, this CAMEO record and this MARPLOT object. When CAMEO wanted to show a given record on the map, or MARPLOT wanted to get linked information about a given object, they would both ask the Linker to get the needed information, launching the required application if necessary. In the actual SPEARS implementation, we did not have the resources to upgrade CAMEO to full Linker-centered operation. So MARPLOT talked to Linker and had it store its links, but CAMEO talked directly to MARPLOT and stored its own links. This worked tolerably well, but the didn't really exploit the Linker concept. In effect, MARPLOT had been split into two parts, a mapping component and a link database component.

With the desire to get rid of two-way links and simplify MARPLOT, and with the complications inherent in the Linker in mind, we invented a simplified linking scheme that we called "hooking." The idea was that MARPLOT would be basically blind to any linking activity. It would be the responsibility of the external database application to keep track of the fact that such-and-such record is linked to such-and-such map object. The application would need to remember the object's ID number, and the name of the map and layer it was on. The application could easily request that MARPLOT show object X on map M, layer L. Going the other way is more problematic: the user

cannot simply select the object in MAPLOT and say "find the associated record" as they could before, since now MARPLOT doesn't know anything about whether the object is linked, and to whom. This is a drawback, but not so serious because applications like CAMEO that link to MARPLOT objects generally install sub-menus in MARPLOT's Sharing menu (the Sharing menu has been around since version 1.0.1). These menus typically include an item named "Get Info." The user can select an object in MARPLOT, and go to the CAMEO:Get Info menu item, at which point CAMEO is invoked and gives information for the selected objects. Thus, the user is required to pick an application to get information from, before requesting information. Our feeling is that this is a relatively small price to pay for the benefits of (1) no two-way links, with all of the associated headache of keeping links consistent and fixing them when they become inconsistent, (2) the possibility of linking many records to a single object, (3) many simplifications to MARPLOT and IAC with MARPLOT in general.

More History: How Applications Greet Each Other

A central design goal has been not to hard-code information about these applications within each other. We have since revised this to say that MARPLOT should not have any hard-coded information about CAMEO, ALOHA and other "clients," but those applications can "know" about MARPLOT, since they specifically use MARPLOT as a tool (this is much the same as saying that your C program can reference the stdio library by name, but that that program shouldn't be required to know specifically about your C program).

In any event, we needed to develop schemes, involving both technical and user interface issues, to allow programs to work with each other in a natural way without knowing about each other ahead of time. For the user interface, we invented Sharing menus. The idea is that each application (MARPPLOT in particular, since it is the "server") has a menu called Sharing. Other application can send MARPLOT the MENU message, which contains the text of a new sub-menu to install in the Sharing menu. When the user chooses an item from, say, CAMEO's Sharing sub-menu in MARPLOT, MARPLOT does not take any action except to send a message to CAMEO (an MHIT message) informing it that item n of its Sharing sub-menu was just selected. CAMEO can then take the appropriate action, usually by initiating another IAC conversation with MARPLOT. There are two key advantages to the use of Sharing sub-menus. First, it allows the user to operate naturally within the server program, while actually performing functions in the client application. For instance, it would be much more awkward if the user had to select the object in MARPLOT, then switch to the CAMEO application and choose a Get Info function from CAMEO's own menu. Second, because applications can save the state of their Sharing menu between runs, and can launch applications as they are needed to respond to Sharing menu selections (when an application installs a Sharing menu in another application, it also provides that other application with the information nessary to launch it in the future), this gives the user the illusion that all of these related applications are always "up and running." Normally, the user would have to explicity start each of the programs that were meant to talk to each other in a given session.

This scheme works well once all of the Sharing menus are installed, but there are some tricky issues about how the Sharing menus get installed in the first place. The basic idea has been for client applciations, when they start, to "broadcast" HOLA (hello) messages to their sever application(s) if they are running. Those server applications respond with OKHI messages, and the client can the send the appropriate Sharing sub-menus with MENU messages. Thus, this scheme requires that the applications get run simultaneously "by hand" just one time, and from then on can launch each other as needed.

To help with this problem, we have introduced the .VWR and .MNU files. A .VWR file contains essentially the same information that is passed in an HOLA or OKHI message, and a .MNU file contains the information passed in a MENU message. When MARPLOT starts up, it looks for .VWR and .MNU files in its "FRIENDS" directory and adds the found menus to its Sharing menu and makes a note of its "friends". This allows an application to "greet" MARPLOT simply by putting a couple of files in MARPLOT's directory.

Dictionary notes

This is version 3.0 of the IAC dictionary. Many of the messages are the same as for MARPLOT 1.0.1, but many have been changed. In general, it is necessary to update old applications in order for them to communicate with MARPLOT 2.x or MARPLOT 3.0. The addition of the VERS parameter in HOLA, OKHI and MENU messages allows MARPLOT to identify older friend applications that are not compatible with the 3.0 dictionary. When MARPLOT hears from such an application, it puts up a note to the user that a newer version of the application is needed, and doesn't add the application as a friend.

=====>	indicates the appropriate response to a message.
#	indicates parameters or options that were not yet implemented as of the date this document was printed

It is not considered an error to send parameters in addition to those required for a certain message. In fact, one way to stay compatible with multiple versions of MARPLOT is to take advantage of parameter name changes by sending parameters under both the new and the old names; newer versions of MARPLOT ignore the old name, and older versions ignore the new names.

Messages From MARPLOT

(all messages include 'SIGN' ('MRP1'), 'PSIG' ('PLOT'), 'MSSG' and 'XTRA' parameters)

Message	Parameters	Description
'BYE '		MARPLOT is quitting. This is to inform you that if you plan to send MARPLOT any more messages, you will have to wait until it gets started again (perhaps by your launching it) and sends you an HOLA message. This message is sent to all friend applications that are currently running.
'CPT!'		Here is the Click Point. You have sent MARPLOT a 'CPT?' message requesting the location of the Click Point. MARPLOT is responding to give you the coordinates. In MARPLOT 1.0.1 western longitudes were given as positive numbers, with eastern longitudes negative. This is the reverse of the standard convention. In newer versions, if you provide the 'VERS' parameter in the 'CPT?' message, the standard conventions are used for the longitude sign in the 'CPT!' message.
	'LAT '	latitude (decimal format)
	'LONG'	longitude (decimal format)

'CTL!'	You have sent MARPLOT a 'CTRL' message, requesting to control the look of objects on a certain map/ layer, and MARPLOT is responding. MARPLOT has written out a file containing the ID numbers of all of the objects on the map/ layer that are inside of or touching the current view rectangle. Each ID number is a 16-digit hexadecimal string. There is no delimiter between the the ID numbers; the 17th character in the file is the first digit of the second ID number. Some of the ID numbers may be all zeros: 0000000000000000. These represent objects that have been deleted or that for some other reason will not be drawn. When writing the TDO file (explained below), you need to include entries for these non-drawn objects; any four bytes will do, since these entries will be skipped by MARPLOT. The PATH parameter is the full path to the file of ID numbers. This file has a name of the form "X.IDS". The task of your application at this point is to read through the IDS file and create a TDO (thematic draw override) file. The name of the file is of the form "X.TDO" (i.e., the same name as the IDS file, but with the TDO suffix) and it goes in the same directory (folder) as the IDS file (i.e., the path stays the same; just the file name changes). The TDO file contains, corresponding to each ID number in the IDS file, five characters, again without any delimiting characters. The four chars specify the look of the object with the given ID number. So the basic routine is to step through the IDS file and for each ID, look up the corresponding record in your database and write five chars into the TDO file. You must write the five chars for each object even if the ID is not found in your database. The meaning of the five chars is as follows: char 1: Color. Use a digit from '1' through 'G' (i.e., "hex" digits), corresponding to the colors in MARPLOT's Color menu. 1 is BLACK, 2 is WHITE, ... , G is YELLOW. char 2: Fill Pattern. Use a digit from '1' to 'A', corresponding to the patterns represented in MARPLOT's Fill Pattern menu. char 3: Line Pattern. Use a digit from '1' to 'A', corresponding to the patterns represented in MARPLOT's Line Pattern menu. char 4: Symbol. Use the ASCII value of the character you want to use from the MARPLOT font (or from the substitute font that is in use for the given layer). char 5: Width. Use a digit from '1' to '6', corresponding to the line widths represented in MARPLOT's Line Width menu. The width value determines the width of lines, as well as the size of the dots when symbols show as dots for the given layer. If any of the five chars is '0', the object will not be displayed at all in MARPLOT. If any of the five chars is 'X', the given attribute will be displayed as it would if the layer were not being controlled. Once you have finished writing the TDO file and have closed it, you must send a DRW+ message to MARPLOT to force it to update its display. MARPLOT will continue to draw using your TDO file until

	'PATH'	full path to IDS file
'HOLA'		Initial greeting from MARPLOT. MARPLOT sends this message to all running friend applications when it starts up. This tells any friend applications that MARPLOT is alive and ready to handle messages.
	'NAME'	"MARPLOT"
	'PATH'	full path to MARPLOT
		empty string; provided for consistency with other applications
		"2"
	=====>	When you get an HOLA message from MARPLOT, you should respond with an OKHI message.
'IMP!'		Report on result of MIE import. You sent an 'IMP2' message to MARPLOT. MARPLOT has attempted to import the specified MIE file.
	'STAT'	"Y" = import was successful, "C" = user canceled import, "E" = error
		NOTE: If the 'IDS' parameter of the IMP2 message was "Y", MARPLOT also writes a file into the same directory as the MIE file to report the ID numbers that were assigned to the objects as they were imported. The name of the file is X.OUT, where the name of the MIE file was X.MIE. X.OUT contains one line for each object ID generated by MARPLOT during the import. Each line has the following format: <app ID> "<MARPLOT ID>" "<layer>" "<map>" <app ID> is the integer identifier that the calling application included at the start of each object line in the extended MIE file, <MARPLOT ID> is the ID number that MARPLOT assigned to the object, <layer> and <map> are the layer and map names of the object. The calling application is responsible for deleting the X.OUT file.
'INFO'		Please show information about this object. The user has selected one or more objects that are linked to your application and has chosen the generic "Get Info" item from the Sharing menu (not the "Get Info" item from your application's Sharing sub-menu). Typically, your application will want to bring itself to the foreground and display information about the selected objects. If you have nothing to display for the selected objects, you should still respond to the user in some way, perhaps by displaying a message in MARPLOT with an ALRT message.
	'LST2'	a return-delimited string where each line (one per object) has the format: <object id> \t <layer name> \t <map name> \t <app ID> <app ID> is the friend-application-owned ID associated with the record at the time it was linked to MARPLOT NOTE: This message is only sent by the SPEARS version of MARPLOT for the Macintosh.

'MAPA'		The user has performed or wants to perform an action on a map that the receiving application owns. LSTA contains the list of objects before the action took place. LSTB contains the same objects after the action took place. In the case of DELETE, which is sent before the objects are actually deleted, LSTB is empty. When you receive a DELETE ACTN, you must decide which if any of the listed objects may be deleted, warn or alert the user if necessary, update your database appropriately, and then send MARPLOT a DELO message, if you actually want to delete some or all of the listed objects. When you receive any other ACTN, just update your database appropriately; no special response to MARPLOT is required.
	'ACTN'	the type of action: one of DELETE, MOVEMAP, MOVELAYER, DRAG, or ADD
	'LSTA'	a return-delimited string where each line (one per object) has the format: <object id> \t <layer name> \t <map name>
	'LSTB'	a return-delimited string where each line (one per object) has the format: <object id> \t <layer name> \t <map name>
		NOTE: In the case that ACTN is DRAG, the LST lines have the format: <object id> \t <layer name> \t <map name> \t <lat> \t <long>
'MHIT'		Your menu item was selected. Once you have installed a menu in MARPLOT's Sharing menu, MARPLOT will send you this message whenever the user selects an item in your menu. You can then take whatever actions are necessary to respond to the menu selection.
	'YRPS'	Pseudo-signature of menu hit; you can ignore this unless you have installed multiple menus in MARPLOT
	'ITEM'	item number hit (the first item is number 1)
	'TEXT'	text of item hit (i.e., the menu item text)
'NBH!'		Neighborhood result. You have sent MARPLOT an NBH? message to create a neighborhood/ threat zone around an existing object, P. MARPLOT is reporting information about the newly created object, N.
	'OBJ2'	a string of the format <object id> \t <layer name> \t <map name>
		NOTE: If OBJ2 is empty, an error ocurred during the creation of the neighborhood object, probably because P was not found.
'OBID'		Here are ID's generated by the most recent old-style import. You have sent MARPLOT an IMPT message to import a list of objects in the 1.0.1 import format. MARPLOT is reporting the ID numbers that were assigned to the imported objects (this includes both newly created ID numbers for new imports and old ID numbers for overwrite imports). Both the 'IDS' and 'LST2' parameters are sent, but an application will typically use just one or the other.
	'IDS '	string of ID's, each (including the last) followed by a return; the order of the ID's is the same as the order of objects in the import file; the string might contain fewer ids than were imported, in case of an error part-way through importing
	'LST2'	a return-delimited string where each line (one per object) has the format: <object id> \t <layer name> \t <map name>

'OKHI'		Acknowledge receipt of HOLA. MARPLOT has received an HOLA message from an application that just started up and is acknowledging so the other application will know MARPLOT is alive. You should treat an incoming OKHI message the same as an incoming HOLA message; they give the same information but you will get one or the other depending on whether your application or MARPLOT is started first.
	'NAME'	alias name of MARPLOT (i.e., "MARPLOT")
	'PATH'	full path to MARPLOT
	'VERS'	"2"
	'DOC '	empty string; included for consistency with other applications
'OVL!'		This layer does or does not exist. If you have sent MARPLOT an OVX? message asking if a particular layer exists. MARPLOT is responding to tell you the answer. If you have sent MARPLOT a MKOV message, MARPLOT is informing you of the layers ID.
	'ANSR'	"Y" if the layer exists, "N" if not
	'OVID'	ID of layer if it exists
#	'ATBS'	attributes of the layer if it exists (see 'SLAT' message for a description of this parameter and the possible values that come with it)
		NOTE: The names 'OVL!' and 'OVID' and historical, from when layers were called overlays.
'USR!'		Here is information about the current MARPLOT user.
	'NAME'	the user's name; for a system not in multi-user mode, this will be User
	'CODE'	the user's 4-character code name; for a system not in multi-user mode, this will be USER
	'PERM'	either EDIT or BROWSE, depending on whether the user has permission to make global changes in MARPLOT
	'PATH'	full path to the user's directory; for a system not in multi-user mode, this will be the MARPLOT directory
'VER!'		Here is MARPLOT's version number.
	'VNUM'	the version number
		NOTE: Do not confuse VNUM with the VERS parameters in other messages; VNUM is MARPLOT's version number while VERS is a shared IAC version number
'VEW!'		Here's the current view.
	'LLAT'	low latitude, as a decimal string (South is negative)
	'LLNG'	low longitude, as a decimal string (West is negative)
	'HLAT'	high latitude, as a decimal string (South is negative)
	'HLNG'	high longitude, as a decimal string (West is negative)

'YROD'	YouR Object Data. You have sent MARPLOT a MYOD or SRCH message requesting information about certain objects. This message contains the requested information. If you specified the "SELECTED" keyword in the OPTN parameter with MYOD, information about all currently selected objects is given. If you specified the "COLLECTED" keyword, information about all objects currently in the search collection is given. If you specified the "IDS" keyword, it is expected that you provided a list of MARPLOT object references in the LST2 parameter. In this case, and according to whether you have included "ANYLAYER", "ANYUMAP, or "ANYMAP" in the OPTN parameter, MARPLOT returns information about all of the objects from the LST2 that it is able to locate. If you have included the "FILE" keyword in the OPTN parameter, the LST2 parameter to MYOD (when it is provided) is taken as the full path to the file containing the LST2 information. Similarly, when "FILE" is specified, the LST2 parameter with YROD is the path name of the file containing the LST2 information. Depending on the OPTN parameter sent with MYOD, the resulting object data contains different information. In all cases information is reported one object per line, with the format <object id> \t <layer name> \t <map name> ... RETURN The "..." in the above line represent extra text that is included with each object, according to the keywords included in the OPTN parameter sent with MYOD. The various keywords, along with the included information, are specified below. The order in which the extra information appears in the text is the same as the order of theentries in the table. OPTN keyword extra info format TYPE \t <type> one of POINT, POLYGON, POLYLINE, RECTANGLE, CIRCLE, TEXT or PICT NAME \t <name> name of the object LATLONG \t <lat> \t <long> decimal strings, S and W negative, center of object # GRAPHICS \t C \t S \t P \t L \t W C = color number, 1 - 16, see MARPLOT menu for values S = symbol, 1 - 255, see MARPLOT menu for values P = fill pattern, 1 - 10, see MARPLOT menu for values L = line pattern, 1 - 10, see MARPLOT menu for values W = line/ dot width, 1 - 6, see MARPLOT menu for values Another OPTN keyword is "TAGS". When specified, the "TAGS"

'NUM '	the number of objects in the LST2
'LST2'	the requested object information
'TYPE'	MYOD is this YROD message is in response to a MYOD message, or SRCH if it is in response to a SRCH message

Messages To MARPLOT

(all messages include 'SIGN', 'PSIG', 'MSSG' and 'XTRA' parameters)

Message	Parameters	Description
'ALRT'		Show alert dialog in MARPLOT with beep. This message causes MARPLOT to beep and to display an alert window with the given text and the given icon in the upper-left corner. The alert has an OK button that the user clicks to dismiss it.
	'TEXT'	text to show
	'ICON'	resource ID of icon to show (0 = stop, 1 = note, 2 = warning)
'BYE '		The friend application is quitting. MARPLOT should not send any more messsages to it.
'CPT?'		Where is the click point? In MARPLOT 1.0.1 western longitudes were given as positive numbers, with eastern longitudes negative. This is the reverse of the standard convention. In newer versions, if you provide the 'VERS' parameter in the 'CPT?' message, the standard conventions are used for the longitude sign in the 'CPT!' message.
	'VERS'	(optional) any value "2" or greater
	=====>	MARPLOT responds with the 'CPT!' message.
'CTRL'		Control the look of objects on a certain map/ layer within the current view. The MODE parameter determines whether you want to start (on) or stop (off) controlling the layer. When you start controlling a layer, it is automatically put into "show" mode. Note that you must specify a map and a layer. Objects on the given layer but on a different map will appear as they would normally, as will objects on the given layer and map, but not in the current view at the time the CTRL message is sent.
	'MAPN'	name of the map
	'LAYR'	name of the layer
	'MODE'	"ON" or "OFF"; ON means you want to start controlling the layer; OFF means you are done controlling the layer. "ON2" is the same as "ON" except the IDS file returned by MARPLOT in the CTL! message will be empty. Use ON2 when you have already computed the TDO file for a given layer (a layer that is pretty certain not to have changed since your last use of it), and you want to avoid the overhead of MARPLOT having to write out the IDS file again. If you use ON2 you must send the VIEW parameter.
	'VIEW'	(send only if MODE is ON2) a string containing four decimal numbers separated by spaces; the numbers represent, in order, low latitude, low longitude, high latitude, high longitude (south and west are negative); the view should be the same view as was shown on the map (as determined using the VEW? message) at the time the layer was originally controlled (MARPLOT needs to know this view to know what the “clipping rectangle” is for the TDO file).
	=====>	MARPLOT writes out an ".IDS" file (which is empty if MODE was ON2) and responds with the 'CTL!' message.

'DELO'		Delete a set of objects. Use this message with caution. You can specify an arbitrary set of objects to be deleted using the LST2 parameter, or delete objects from a single layer using the LAYR parameter. In the latter case, you can use the ' IDS ' and 'MAP ' parameters to delete only certain objects on the layer.
	'LST2'	(must be given if LAYR is not given) a return-delimited string where each line (one per object) has the format: <object id> \t <layer name> \t <map name> for a large number of objects, as an alternative to sending a large LST2 parameter, the LST2 information can be written to a file; in this case, the LST2 parameter should be the word FILE, followed by a tab, followed by the full path to the file, that is FILE\t <path> MARPLOT does not delete the file after reading it
	'IDS '	(must be given if LST2 is not given) return delimited and return-terminated list of ID's of objects on the named layer to be deleted
	'LAYR'	(must be given if LST2 is not given) the name of the layer on which the objects are to be found
	'MAP '	(optional) the name of the map from which objects on the named layer are to be deleted (objects on the layer but on other maps will not be touched); use the name "USER" to indicate the current user map. Default is the the current user map.
'DLOV'		This message is used to delete all the objects from a layer; unless the REMOVELAYER option is used, the layer itself is still retained in MARPLOT, in its specified position in the layer list.
	'LAYR'	The name of the layer
	'MAP '	(optional) The name of the map from which the named layer is to be deleted (objects on the layer but on other maps will not be touched); use the name "USER" to indicate the current user map.;use the name ALLMAPS to delete the layer on all maps.;Default is the the current user map.
	'OPTN'	(optional) a string containing keywords from the following table: <u>keyword</u> — <u>meaning</u> REMOVELAYER — this keyword requests that the the layer be deleted from MARPLOT's layer list, after the specified layer map combination specifed is deleted; Note: the layer will not be deleted from the layer list if there are objects remaining on this layer on another map
'DRW+'		Re-enable map window updates, and update the map window.

'FRWD'		There are a number of technical issues involved in getting an application to come automatically to the foreground. These issues are different for each platform/ system. In some cases, when an application wants to bring some application (often itself) to the foreground, it is easier (and sometimes more polite) to ask another application to do the job. Your application can ask MARPLOT to bring it to the foreground using this message.
	'WHO '	signature of application to bring forward (usually the sending app itself)
	'NAME'	name of application to bring forward; on Windows, NAME should be the title of your main window, or the name of your main window class.
		NOTE: On the Macintosh, it is sometimes better to use the Notification Manager and let the user bring you forward.
'GOTO'		Set the Focus Point at this location and center on it, using this scale.
	'LAT '	latitude value, as decimal string (South is negative)
	'LONG'	longitude value, as decimal string (West is negative)
	'SCAL'	(optional) n, where desired scale is 1:n (if not given, scale is not changed)
'HOLA'		Initial greeting from a friend application. The friend sends this message to MARPLOT when it starts up and sees that MARPLOT is running. This tells MARPLOT that the friend is alive and ready to handle messages.
	'VERS'	"2" or greater; needed to show MARPLOT you are using the updated IAC messages
	'NAME'	name of the friend application
	'PATH'	full path to friend applications's executable file
	'DOC '	(optional) full path to default document to be opened when MARPLOT launches the friend
'IMP2'		Import this MIE file.
	'FILE'	full path of MIE file to import
	'IDS '	"Y" = return a list of the IDs assigned to the imported objects, "N" = no list (default)
		NOTE: When the 'IDS ' parameter is "Y", the MIE file is expected to be in a slightly modified format, where each object is preceded by an integer chosen by the calling application. When MARPLOT writes out the file of generated IDs, each ID is preceded by its object number.
	=====>	MARPLOT responds with an 'IMP!' message.

'IMPT'		Old-style import (MARPLOT 1.0.1 format).
	'TEXT'	text of "file" to be imported; instead of importing from an actual file on disk, you simply send the text of the old-style import file with your message by using this parameter; see the MARPLOT 1.0.1 import format for the format of an old-style import file
	'MAP '	(optional) the name of the map onto which the objects should be imported; if this parameter is not specified, the current "user's map" is used
	'MOD?'	(optional) "R" (default) = if an ID in the import file matches the ID of an object on the given layer, overwrite the object with the import file data; if there is no object with a matching ID, create a new object with the ID "M" = if an ID in the import file matches the ID of an object on the given layer, overwrite the object with the import file data; if there is no object with a matching ID, do NOT create a new object with the ID "N" = ignore the object ID's in the import file and generate a new object with a new ID for each import file entry
	'SEL?'	(optional) "Y" (default) = when objects have been imported, select only imported objects in MARPLOT "N" = don't change selections after import, don't select imported objects
	'SHOW'	(optional) "Y" (default) = bring MARPLOT forward once the import is complete and change the view if necessary so that all imported objects are visible "N" = do not bring MARPLOT forward and do not change the view
	'DIST'	(optional) "Y" = interpret all coordinate values in the import file not as absolute positions, but as distance offsets from the point given in the CNTR parameter "N" (default) = treat coordinate values in the import file normally as absolute lat/ long values
	'CNTR'	this is a string of the format "lat, long", where lat and long are decimals; this parameter must be provided if the DIST parameter is "Y"; these lat/ long values define the point from which all coordinate values in the import file are interpreted as offsets

	'UNIT'	(optional) this is only used when the DIST parameter is "Y"; it specifies the the units that the offset values in the import file represent "M" (default) = miles "Y" = yards "K" = kilometers "E" = meters
	=====>	MARPLOT responds to an IMPT message with an OBID message in order to tell the sending application the ID's of the objects that were created and/ or overwritten as a result of the import.
		NOTE: All longitude values in this file are oriented in the reverse from the normal convention: increasing positive to the West and decreasing negative to the East.

'LGND'		Define and show a legend on the map. The legend can be specified as a bitmap (picture) file with the PATH parameter or as a list of names and attributes with the LIST parameter. The legend temporarily overrides any legend the user is currently using. Send a LGND message with no parameters to remove the previously-sent legend and revert back to the user's legend (if any).
	'PATH'	full path to the picture (or bitmap) file to be used as a legend; this file should not already be in the MARPLOT directory; the file is not deleted after it is copied to the MARPLOT directory by MARPLOT
	'LIST'	return-delimited and return-terminated list of lines for the legend; each line contains the name to be shown on the list, a tab, and then five characters that specify the graphical attributes to be shown on the line; the five characters specify color, fill pattern, line pattern, symbol and line width, as explained in the CTL! message but there is also an expanded format to allow RGB colors as explained below; the symbol characters corresponding to ASCII values 33 and 34 (decimal) are special: 33 means to show just a polyline graphic using the given color, fill pattern, line pattern, and width, while 34 means to show just a polygon graphic using the given color, fill pattern, line pattern, and width; for all other ASCII values, the corresponding symbol is shown in the given color (and the fill pattern, line pattern, and width are not used) note: if the user is showing a layer-list legend at the time the LGND message is received, the layer-list lines are retained at the bottom of the legend, below the lines specified in LIST RGB extension: The first 5 characters continue to represent the color, fill pattern, line pattern, symbol and line width, but if the color char is an 'R', then the color is an RGB color specified by 9 additional characters following the original 5, specifying the red, green and blue values on a scale of 0 to 255.
	'XOUT'	(optional) return-delimited and return-terminated list of names of layers NOT to be included in the legend when the user is showing a layer-list legend at the time the LGND message is received; that is, these specific layers are not retained in the new legend, but other layers that the user was showing in the legend are retained

'MAP '		Use this map. This message adds a map to MARPLOT's map list and by default considers the map to be "owned" by the sending application. If the map is already in the map list, it changes to "owned" status. MARPLOT does not allow the user to delete layers if they contain objects on owned maps. Owned maps can be removed by the user if the owning application is not running. MARPLOT does not allow the user to rename owned maps. New feature:You can specify a parameter to indicate that your application does not wish to be the owner. Using this parameter allows your application to add maps to MARPLOT's map list without becoming the owner.
	'PATH'	full path name to map folder, including the ':' or '\' terminator

	'OPTN'	a string containing keywords from the following table: keyword — meaning DEFAULTMAP — the specified map becomes the default map (when the user creates an object, it will go on this map by default) NODEFAULT — (default) the specified map does not become the default DELETENO — (default) objects on this layer may not be deleted DELETEYES or DELETEOK — objects on this layer may be freely deleted by the user DELETEALERT — when the user attempts to delete objects on this layer, MARPLOT sends a MAPA message to the owning application (the owning application must be running BEFORE the user makes the change) MOVEMAPNO — (default) objects on this map may not be moved to other maps MOVEMAPYES or MOVEMAPOK — objects on this map may be freely moved to other maps by the user MOVEMAPALERT — when the user moves an object on this map to another map, MARPLOT sends a MAPA message to the owning application (the owning application must be running BEFORE the user makes the change) MOVELAYERNO — (default) the user cannot change the layer of objects on this map MOVELAYERYES or MOVELAYEROK — the user can freely change the layer of objects on this map MOVELAYERALERT — when the user changes the layer of objects on this map, MARPLOT sends a MAPA message to the owning application (the owning application must be running BEFORE the user makes the change) DRAGNO — (default) the user cannot change the position of objects on this map DRAGYES or DRAGOK — the user can freely change the position of objects on this map DRAGALERT — when the user changes the position of objects on this map, MARPLOT sends a MAPA message to the owning application (the owning application must be running BEFORE the user makes the change) ADDNO — the user cannot add objects to this map, either by using tools or moving them from other maps ADDYES or ADDOK — (default) the user can add objects to this map ADDALERT — when the user adds objects to this map, MARPLOT sends a MAPA message (the owning application must be running BEFORE the user makes the change) OWNEDNO — add the map to MARPLOT's map list but do NOT consider the map to be "owned" by the sending application.

'MENU'		Install sub-menu in MARPLOT's Sharing menu. This adds a new sub-menu to MARPLOT's Sharing menu. If a sub-menu with the same name already exists, it is replaced with the new menu. (Thus, a friend application can send a MENU message each time it greets MARPLOT, without worrying about duplicating the menu.) Menus installed in MARPLOT are automatically saved by MARPLOT. They can be used later, even when the friend application is not running. In this case, MARPLOT launches the friend application before sending it the MHIT message.
	'VERS'	"2" or greater; needed to show MARPLOT you are using the updated IAC messages
	'NAME'	name of menu
	'ITMS'	return-delimited string of menu items
	'PATH'	full path to the friend application's executable file
	'DOC '	(optional) full path to default document to be opened when the friend is launched
		NOTE: If for some reason the MARPLOT user wants to remove a sub-menu from the Sharing menu, the user can delete the appropriate ".MNU" file from the MARPLOT "FRIENDS" directory.
'MKOV'		Make layer. This message asks MARPLOT to create a new layer. This will not create a new layer if one with the same name already exists.
	'NAME'	name of overlay to create
	'ATBS'	attributes of the new layer (see 'SLAT" message for a description of this parameter and the possible values that come with it)
	=====>	MARPLOT responds with an 'OVL!' message to communicate the layer ID of the newly created layer.
	NOTE:	This message should be called MKLR but is misnamed to support existing applications.

'MYOD'	MY Object Data. This message is used to request information about one or more objects in MARPLOT. The OPTN parameter is used to specify which objects you would like information about, and what types of information you want for the chosen objects. OPTN is a string containing any number of keywords separated by spaces, tabs or commas. The meanings of the various keywords possible in the OPTN string are given in the following table and are explained more fully in the documentation for the 'YROD' message. keyword — meaning The following three keywords are mutually exclusive. SELECTED — give info for the objects currently selected in MARPLOT COLLECTED — give info for the objects currently in the search collection IDS — give info for the specific objects specified in the LST2 parameter FILE — write the data to a file and pass its path in the LST2 parameter from YROD; also, if "IDS" is specified and "FILE" is also specified, the "LST2" parameter is taken as a path to a file containing the LST2 information The following three keywords only have meaning when used with "IDS". ANYLAYER — if an object is not found on the given layer, search all layers ANYMAP — if an object is not found on the given map, search all maps ANYUMAP — if an object is not found on the given map, search all "universal" maps MIE — write the data using the full MIE format (using one line per object) The remaining keywords cannot be used with the "MIE" keyword # TAGS prefix each data element with a tag TYPE — include the type of the objects in the data NAME — include the name of the objects in the data LATLONG — include the lat/ long of the centers of the objects in the data # GRAPHICS — include the graphical attributes of the objects in the data

	'OPTN'	a string containing keywords from the table above
	'LST2'	(only necessary when the "IDS" keyword is specified in OPTN) a return-delimited string where each line (one per object) has the format: <object id> \t <layer name> \t <map name> if you have specified ANYLAYER in OPTN, you can leave the <layer name> empty and MARPLOT will search all layers; similarly, if you have specified ANYMAP or ANYUMAP, you can leave the <map name> blank and MARPLOT will search; if "FILE" is specified in OPTN, LST2 is the full path to the file containing the information
	=====>	MARPLOT responds with the YROD message.
'NAME'		Set the name of an object. Note: The object's name on the screen only changes visually when MARPLOT is the front application (or when it later becomes the front application after a NAME message). When using NAME, you should either be sure MARPLOT is or is about to become the front application, or else use a DRW+ message to force an update in the background.
	'OBJ '	a string of the format <object id> \t <layer name> \t <map name>
	'NAME'	the new name for the object
	'PFIX'	(optional) the prefix of the name (such as "N." or "NW"); if the prefix is included in this parameter, it should not also be part of NAME
	'ID '	(optional) a 16-characater (hex) ID number to replace the object's current ID number

'NBH?'		Make a neighborhood/ threat zone about a symbol or polyline object by creating a circle or a polygon made up of many pieces. This message is useful for friend applications that want to indicate the area on a map within a given distance from a given object. For instance, if the object is a polyline representing a transportation route, this message could be used create a threat zone "tube" with a certain radius about the polyline to show the area that would be affected by a spill of a certain chemical anywhere along the route. The neighborhood about a polyline is constructructed from several polygon pieces: each vertex of the polyline is surrounded by a circular polygon piece and each segment is surrounded by a four-sided polygon piece that forms a box about it. The union of all of these pieces in the single threat zone object covers the area of the map within the specified distance from any point on the polyline. For the purposes of the following parameter descriptions, let P be the polyline or symbol object about which the neighborhood is to be built, and let N be the neighborhood object.
	'OBJ1'	information about P in a string of the form <object id> \t <layer name> \t <map name>
	'LYR2'	the name of the layer on which N is to be created
	'MAP2'	(optional) the name of the map on which N is to be created (default = user's map)
	'ID2 '	(optional) the id number of N (default = a MARPLOT-generated ID)
	'RAD '	radius of N in miles
	'NAME'	(optional) name assigned to N (default = "")
	'SHOW'	(optional) "Y" (default) = bring MARPLOT forward once N is created and change the view if necessary so that N is visible "N" = do not bring MARPLOT forward and do not change the view
	=====>	MARPLOT responds with a 'NBH!' message.
'OKHI'		Acknowledge receipt of HOLA from MARPLOT. The friend application has received an HOLA message from MARPLOT, and is acknowledging so that MARPLOT will know the friend is running.
	'VERS'	"2" or greater; needed to show MARPLOT you are using the updated IAC messages
	'NAME'	name of the friend application
	'PATH'	full path to the friend application's executable file
	'DOC '	(optional) full path to default document to be opened when MARPLOT launches the friend
'OVX?'		Does this layer exist?
	'NAME'	name of layer
	=====>	MARPLOT responds with an 'OVL!' message.

'SHOW'		Show certain objects in MARPLOT, either on the map or in the search collection.
	'LST2'	a return-delimited string where each line (one per object) has the format: <object id> \t <layer name> \t <map name> <map name> (in this message and in other messages with LST2 parameters) can be either the actual name of the map, or the (5) digits of the map's default location code (which usually corresponds to a TIGER state+count) if you have specified ANYLAYER in OPTN, you can leave the <layer name> empty and MARPLOT will search all layers; similarly, if you have specified ANYMAP or ANYUMAP, you can leave the <map name> blank and MARPLOT will search
	'FILE'	(optional) full path to file containing LST2 information; in this case you should not include a LIST2 parameter

'OPTN'	a string of keywords separated by spaces, tabs or commas; the meanings of the various keywords are given in the following table: <u>keyword</u> — <u>meaning</u> >>>>>> The following two keywords are mutually exclusive. DRAW — show the given objects on the map; this is the default if no keyword is given # COLLECTION — put the given objects in the search collection <<<<<<<<<<<<<<<<<<<<<<<<<<<< ANYLAYER — if an object is not found on the given layer, search all layers ANYMAP — if an object is not found on the given map, search all maps ANYUMAP — if an object is not found on the given map, search all "universal" maps >>>>>> The following four keywords are mutually exclusive and only have meaning if the DRAW keyword is given YESZOOM — change the scale of the map to cover just enough area to show the objects NOZOOM — leave the scale of the map unchanged, except possibly zoom out to encompass all shown objects ; this is the default if no keyword is given NEVERZOOM — leave the scale of the map unchanged, even if it means leaving some shown objects off the screen SAMEVIEW — leave the current view unchanged, even if it means leaving some shown objects off the screen <<<<<<<<<<<<<<<<<<<<<<<<<<< >>>>>> The following two keywords are mutually exclusive and only have meaning if the DRAW keyword is given. YESSELECT — select the shown objects, deslecting any others; this keyword is equivalent to the old keyword SELECT which is still supported NOSELECT — do not select or deselect any objects; this is the default if no keyword is specified <<<<<<<<<<<<<<<<<<<<<<<<<<< CTRL — take control of the layer map these objects are on and hide all objects not in the list.of objects in this message; this keyword will be ignored unless the objects are all on a single layer on a single map; using this keyword here is similar to the CTRL message, but the bounding rect is the entire world; calling program is responsible for turning the control off; this only has meaning if the DRAW keyword is given

		NOTE: When the objects are put in the search collection, and the sending application is a known friend, MARPLOT changes the title of the Search Collection dialog box to "Search Collection From N", where N is the name of the friend application.
'SLAT'		Set the attributes of a layer.
	'NAME'	the name of the layer
	'ATBS'	a string containing any of the following keywords, plus their associated data, separated tabs or commas; if a keyword has an "extra data" entry, it is expected that that data is the next item or items in the ATBS string. keyword — meaning [extra data] LOCKEDYES — layer is locked LOCKEDNO or UNLOCKED — layer is unlocked TEMPORARY — layer is temporary PERMANENT — layer is permanent APPOWNED — layer is owned by sending app MARPLOTOWNED — layer is owned by MARPLOT DELETEOK — even if layer is owned, user can delete objects on layer DELETENO — when layer is owned, user cannot delete objects on layer # GRAPHICS — default layer graphics [C, S, P, L, W (see YROD)] INDIVIDUAL — individual graphics COMMON — use default graphics HIVISSCALE — scale at which layer shows [n, where scale is 1:n] LOVISSCALE — scale at which layer hides [n, where scale is 1:n] DOTSSCALE — scale at which symbols->dots [n, where scale is 1:n] NAMESSCALE — scale at which names show [n, where scale is 1:n] SHOWNAMESMODE — show + names mode SHOWMODE — show mode RANGEMODE — range mode HIDEMODE — hide mode # NAME — name of layer # POSITION — position of layer in list ["TOP", "BOTTOM" or n (1 = top)]

'SRCH'		Search for objects.
	'FUNC'	a keyword specifying the type of search you want to perform keyword — meaning ANYNAME — show the given objects on the map NAMESTARTS — names that start with... NAMECONTAINS — names that contain... WITHIN — objects within a certain distance of... NOTWITHIN — objects not withing a certain distance of... TOUCHING — objects that touch.. NOTTOUCHING objects not touching... NOTE: To have the found objects in the resulting YROD message sent in a file, the recommended method is to use the OPTN parameter described below., but you can also use the older method of appending a space and the word FILE to the FUNC parameter. For example: WITHIN FILE In this case, the LST2 parameter with YROD is the path name of the file containing the LST2 information (i.e., the found objects).
	'NAME'	the text to match for a NAMESTARTS or NAMECONTAINS search
	'DIST'	the distance for a WITHIN or NOTWITHIN search Required for WITHIN and NOTWITHIN searches.
	'UNIT'	(optional; default = MI) one of the following keywords, specifying the units of the values in the DIST parameter: FT, YDS, M, KM, MI, NM.
	'OF '	(optional; default = FOCUSPOINT) referent for a WITHIN , NOTWITHIN, TOUCHING or NOTTOUCHING search; one of the following keywords: FOCUSPOINT, MARKEDPOINT, SELECTEDOBJECTS, PREVCOLLECTION.
	'LYRS'	a return-delimited list of the layers to be searched; an empty string indicates that all layers should be searched. Note that unless the SEARCHSIMILARLAYERS option key is used, layers must be explicitly listed. For example, if the parameter is "Roads ", the "Roads" layer will be searched but the "Roads (Major)" layer will not be searched. If you want MARPLOT to search "Roads" and all of the similar layers, use the SEARCHSIMILARLAYERS key in the OPTN parameter.
	'MAPS'	a return-delimited list of the maps to be searched; an empty string indicates that all maps in the current view should be searched; # the special value SEARCHALLMAPS indicates that all maps should be searched

'OPTN'	The OPTN parameter is used to specify what to do with the found set of objects and what types of information you want returned for the found objects. OPTN is a string containing any number of keywords separated by spaces, tabs or commas. The meanings of the various keywords possible in the OPTN string are given in the following table and are similar to those in the 'MYOD'" message. keyword — meaning SEARCHSIMILARLAYERS — search layers with names similar to the names in the LYRS parameter. For example, if the LYRS parameter is "Roads ", both the "Roads" layer and the "Roads (Major)" layer will be searched when this keyword is used. SHOWINCOLLECTION — show the found objects in the MARPLOT search results dialog. MARPLOT will come forward, perform the search and present the user with the results. Note that the search results are not returned via a YROD message. The remaining keywords apply to the returnedYROD message and have no meaning when used with the "SHOWINCOLLECTION" keyword. FILE — write the data to a file and pass its path in the LST2 parameter from YROD. Depending on the keys sent in the OPTN parameter the resulting object data contains different information. In all cases information is reported one object per line. The default format is <object id> \t <layer name> \t <map name> ... RETURN The following keywords can be used. MIE — write the data using the full MIE format (using one line per object) The remaining keywords cannot be used with the "MIE" keyword # TAGS prefix each data element with a tag TYPE — include the type of the objects in the data NAME — include the name of the objects in the data LATLONG — include the lat/ long of the centers of the objects in the data # GRAPHICS — include the graphical attributes of the objects in the data

	=====>	Unless the keyword SHOWCOLLECTION is used, MARPLOT responds with a YROD message, with SRCH in the TYPE parameter. The LST2 parameter of the YROD message contains the results of the search.
'TRNG'		Put MARPLOT into training mode. This simply removes, for the remainder of the present session, all of the maps in the map list. It adds the map given in PATH as the one map in the list, which is treated as the User's map. When in training mode, the Map List dialog box is unavailable.
	'PATH'	full path to the new User's map, including final directory delimiter but not including the NAME.MAP file name
'USR?'		Who is the current MARPLOT user?
	=====>	MARPLOT responds with a 'USR!' message.
'VER?'		What is MARPLOT's version number?
	=====>	MARPLOT responds with a 'VER!' message.
'VEW?'		What is the current view?
	=====>	MARPLOT responds with a 'VEW!' message.
'VIEW'		Show this world rect on the map.
	'LLAT'	low latitude, as a decimal string (south is negative)
	'LLNG'	low longitude, as a decimal string (west is negative)
	'HLAT'	high latitude, as a decimal string (south is negative)
	'HLNG'	low longitude, as a decimal string (west is negative)

Messages specific to ALOHA

'MAPA'		I'm the mapping application. By default, MARPLOT is the mapping application with which ALOHA communicates. If another mapping application wants to take over that role, and is capable of supporting all of the MARPLOT-ALOHA IAC communication, it should send ALOHA this message when it starts up and/ or detects that ALOHA is running. ALOHA then uses the signature and pseudo-signature of the sending application for all mapping IAC.
	'SIGN'	the signature of the sending application
	'PSIG'	the pseudo-signature of the sending application
	'PATH'	the full path to the sending application's executable file
	'DOC '	the full path to the sending application's default document (or empty)
	'SAVE'	"YES" means that ALOHA should record the sender as the permanent mapping application; "NO" means that ALOHA should use this sender for this session only and go back to MARPLOT for the next session; "ON" means to use this mapping application now during this session; "OFF" means to resume using MARPLOT now during this session.