MARPLOT® 4.0 Technical Documentation

For Overlay Objects and Inter-application communication

Table of Contents

Overlay Object Import/Export

- MARPLOT Simple Point Format
- MARPLOT Import/Export (MIE) File Format
- MARPLOT 1.0.1 Import/Export File Format

MARPLOT Simple Point Format

For Overlay Objects

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 <> overlay 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, overlay, 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 overlay or map. In this case you would use the placeholder value 0 (zero) for both the overlay and map name fields.

This table shows the type of data in each 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
overlay name	string of characters / My Sites	the name of the topmost overlay	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 symbol in the MARPLOT font set 1 (or 256 plus the ASCII value of the symbol in font set 2)	LANDMARK	0
color	one of the names of the available colors (listed in MARPLOT Color menu) or an RGB value / R123G64B0	BLACK	0
ID	a 16-digit number (the digits are 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 overlay of a given map (i.e., no two objects on a given overlay of a given map should have the same ID)	a random code	N/A

MARPLOT Import/Export (MIE) File Format

For Overlay Objects

```
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 overlay 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

overlay = 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 used 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 "overlay 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 "overlay name" "<map name>" POINT "<id number>" 0 "X00" 0 ONLY 0
BLACK 1 LANDMARK -76.992700 38.844000
```

Notes: Items in <brackets> can be empty (i.e., ""). The constant LANDMARK can be replaced with any integer ASCII value to specify any character in the MARPLOT font set 1 (or 256 plus the ASCII value to specify characters in font set 2).

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

For Overlay Objects

Note: While this old format is still supported, we recommend that you use the MIE file 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 IDs 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 determines 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 northern- 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> value is no longer used. However, you must have a value (e.g., -1) in the file.

g. The <size> value is no longer used. However, you must have a value (e.g., -1) in the file.

h. The <style> value is no longer used. However, you must have a value (e.g., -1) in the file.

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 = horizontal 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 = horizontal 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 set 1 (or 256 plus the ASCII number of the character in font set 2) that is drawn to represent the object. You can determine these numbers by looking at the MARPLOT helps. 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. Next are the pairs, again using decimals with western longitudes positive. Even if there are many points, they must all be on this one line of the file. To indicate that the polygon is "closed", the coordinates of the first point should be the same as the coordinates 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 line containing the polygon points. On the other hand, if you do specify <type> 6, you MUST include this line.

The next seven lines of the object definition were used to define link information between the object and an external application. This feature is no longer supported, but the fields must be present. These lines should be blank.

Example:

This sample import file contains two objects, each on a different overlay. 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




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
```

Map File Formats

The Map file formats are cross platform (Macintosh and Windows) and so it is important to understand the binary format and the extended ASCII character set being used in these files.

Binary format

In short and long integers, the most significant byte is on the left (old Mac binary style, opposite of Intel style).

The Extended ASCII Character Set

The following 16 characters from the ISO 8859-1 may appear in the Census 2000 TIGER/Line ® files:

Character Name ISO (dec, hex)

Á A-Acute Accent 193,c1
á a-Acute Accent 225,e1
É E-Acute Accent 201,c9
é e-Acute Accent 233,e9
Í I-Acute Accent 205,cd
í i-Acute Accent 237,ed
Ñ N-Tilde 209,d1
ñ n-Tilde 241,f1
Ó O-Acute Accent 211,d3
ó o-Acute Accent 243,f3
Ú U-Acute Accent 218,da
ú u-Acute Accent 250,fa
Ü U-Diaresis 220,dc
ü u-Diaresis 252,fc
Å A Ring 197,c5
å a Ring 229,e5

ÁáÉéÍíÑñÓóÚúÜüÅå

It was originally specified that the .LYR files would be cross-platform and to that end, those files used MAC binary. Unfortunately, the extended char set was not specified at that time.

Starting with MARPLOT 3.3, the TIGER files have started using the extended ASCII set for a number of Spanish characters; we need to address the problem of the assumed character mapping for the names/text in the “binary” files (.LYR, .SUM etc.).

The basic plan is that MARPLOT will assume the “binary” files are using the “Windows” character set. Specifically the char set used by Census (ISO Latin).

The MAC will convert that char set when writing and reading the "binary" files, just as the Windows MARPLOT converts the MAC binary in these files when writing and reading.

The final catch is the import/export files. Because we wanted the users to be able to export objects and then hand edit them, it makes sense that the export files use the char set that is the native char set of the platform. Fortunately the Census only uses a limited number of the extended characters, and those extended characters are non-overlapping in the MAC and Windows extended char sets, so we can, in theory, detect which is being used and handle it accordingly.

MARPLOT 3.3 includes code that looks at each "name" and determines which character set is being used on a name by name basis. So in theory, a MAC MIE file can be imported using a Windows MARPLOT, and the conversion will be automatic.

Note under this implementation, you could actually switch character sets in the middle of an MIE file (not a planned-for feature, but the current implementation allows it).

Auto Character Set Recognition

Here is how the new auto character set recognition code works:

- Given a character string, the characters of the string are examined to see if they are all in the supported extended ASCII set for the given platform.
- If they are, the string is left untouched.
- If they are not, and if all of the characters are in the character set of the other platform, the string is converted.
- If the characters of the string do not fit into the supported extended ASCII set of either platform, the string is left untouched.

The auto character set recognition feature of MARPLOT can be turned off by including the line
`UseAutomaticCharacterSetRecognition NO`
in the MPConfig.txt file.

Users may want to do this if they are using, say, the French character set which includes characters outside of the supported extended ASCII set.

The "User's Map" Problem

The smart apostrophe was used in "User's Map" on the Macintosh in previous versions of MARPLOT. The MAC smart apostrophe collides with the Windows extended character set.

The new MARPLOT standardizes on the normal apostrophe on both platforms, but it has to deal with the historical files. Because of the historical files, the new MARPLOT will inspect map names before using auto character set recognition. If the map name is User's Map, the map name will be changed to User's Map.

Levels of support

- Extended ASCII object names (and prefixes)
- Extended ASCII overlay and map names
- Extended ASCII group names
- Extended ASCII view names

(General files)
The following files are “binary” and use the Windows char set:

- USERS.PLT
- SETTINGS2.PLT
- LAYERS2.PLT
- GROUPS.PLT
- extra maps.PLT (**Note:** path names will not be cross platform anyway)
- Views

The following files are text files and it is possible that users will be manipulating them, so we will write these files using the native platform char set and use the auto_char_set_recognition technique when reading these files:

- Saved search collections
- CDMaps/ MARPLOT.VNX

(map/overlay files)
The following files are “binary” and so use the Windows char set:

- .SUM (the first 4 chars of the name)
- .SM2
- .LYR (object name, object prefix., object text)
- Name.Map

These files are assumed to be in native platform char set:

- Font name --
- Meta data -- (not an issue)

(Import export files)
The import/export files are text files and it is possible that users will be manipulating them. So MARPLOT will write them using the native platform char set.

The auto_char_set_recognition technique is used when reading the following files:

- .MIE
- Simple point

The following files assume the platform native char set (i.e., auto_char_set_recognition technique is not supported):

- old MARPLOT IMPORT format

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 OBJ file (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 overlay 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.

LYR (overlay information) files

overlay name (32 chars)
overlay'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 overlay 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 overlay on this map (long)

FNT (alternate font) files

This feature of earlier version of MARPLOT has been removed for MARPLOT 4.0.

VEW (view information) files

view name (32 chars)
map name (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)
file name (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>" "<overlay name>" "<id>" <flags> <offset>

Where

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

Note: <offset> is currently unused; for each object, MARPLOT searches for the given <id> on the given overlay 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 CAMEOfm.VWR:

```
CFAM
CFAM
CAMEOfm
C:\Program Files\CAMEO\CAMEOfm.EXE
C:\Program Files\CAMEO\MapData.CAM
```

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.

CDMAPS and volume index (.VNX) files

Note: This feature is still supported in MARPLOT 4.0, but the Basemap Builder has replaced the needs for large collections of maps.

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.

There are two methods supported. First MARPLOT examines all .TXT files in the CDMaps folder. Then it looks at the root level of all mounted volumes/drives for a file called MARPLOT.VNX.

The format of these volume index files is as follows:

The first line gives the volume label (disc name) of the CD or drive containing the maps. In the case where the file is being used to index a hard drive, this can be a drive letter followed by a colon (e.g., "C:").

This is followed by n lines, one for each overlay that is represented in any of the maps on the CD. Each line has the following format:

```
N FILENAME, Overlay_Name
```

where `N` is a unique character, such as a digit, used to flag the overlay throughout the file, `FILENAME` is the name of the files, without the suffix, that the overlay is stored in on the disc, and `Overlay_Name` is the name of the overlay as it appears in MARPLOT. The comma is the delimiter, hence both `FILENAME` and `Overlay_Name` can contain spaces.

Following these overlay 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. Prior to MARPLOT 3.3.1, the `path` and `FIPS` could not contain any space characters, since the delimiter is a space.

Starting with MAPLOT 3.3.1, if this line contains two tab characters, the line is assumed to be tab delimited. This allows the use of spaces in the path and FIPS code.

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

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

where is N is the unique character corresponding to the overlay, 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 overlay on the map, NUMOBJECTS is the number of objects in the overlay on the map. To convert to the MARLOT format 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
.
.
```

Overlay index (.LNX) and group index (.GNX) files

New to MARPLOT 3.3 is the idea of overlay and group index files to augment the volume index (.VNX) files.

Whenever MARPLOT processes a volume index (.VNX) file or a CDMaps (.TXT) file, it will look for a file in the same location with the same base name and .LNX and .GNX extensions.

The .LNX and .GNX files contain information about the desired default setting of the overlays and the overlay groups that should be used for the overlays described in the corresponding .VNX or .TXT file.

The .LNX file is simply a renamed LAYERS2.PLT file and the .GNX file is simply a renamed GROUPS.PLT file.

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.

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.

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 = "00000Bssccppppp"

  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 counterpart.

- 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)

MARPLOT Colors

Millions of Colors

With the release of MARPLOT 3.2, MARPLOT allows each object to carries 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 MIE and Simple Point MARPLOT import formats 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). Starting with MARPLOT 3.3.3, MARPLOT will use the RGB format in export files for those objects not using the standard 16 colors.

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.

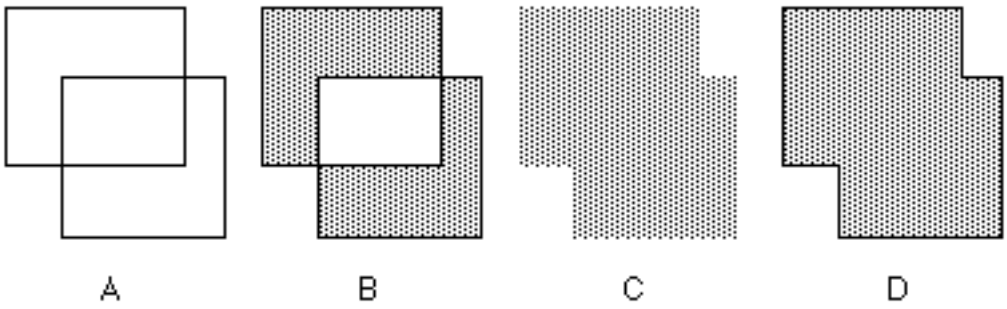

Figure A above shows two overlapping polygons. When the union of these two polygons is computed, we would like to get the polygon in figure D. In most cases, it is not difficult to compute results as in figure D. However, in the case of very convoluted polygons, and especially polygons that have one 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 are reported incorrectly by MARPLOT 3.0, and not at all by MARPLOT 3.2.

Inter-application Communication (IAC) Dictionary

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.

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 suite 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, CAMEOfm's signature is CFAM, and ALOHA's signature is ALH5. On Windows, the signatures are used as the server name of the application.

To allow for greater flexibility, and to accommodate some quirks of the Macintosh, 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, CAMEOfm's pseudo-signature is CFAM, 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

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 receive 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 parameters, 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

To start:

You may either assume that the NOAA_32.DLL is in the Windows 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 Windows directory, and replace it if so.

Load the NOAA32 DLL that is in the Windows directory 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 = 0;
char gMySignatureStr[] = "MySg"; // a 4 character identifier you wish to use
for your application
HWND gMyMainWindowHWND = 0;
char gMyMainWindowClassName[256];

long MyStartupTasks(HWND myMainWindowHWND, char * myMainWindowClassName) //
call this when you program is starting
{
      long errorCode = 0;
      gMyMainWindowHWND = myMainWindowHWND;  // record the value of your
main window handle
      if(myMainWindowClassName)
strcpy(gMyMainWindowClassName,myMainWindowClassName);

      return errorCode;
}

void MyShutdownTasks(void)
```

```
{
     // say goodbye to ALOHA if it is running

     CallNEBye(); // this will unload the NOAA 32 dll
}

//////////////////////
//////////////////////


void LoadNoaaDll (void)
{
   char dllPath[256];

   if(gNoaaDllInst)
      return;  //if already loaded, don't reload

  GetWindowsDirectory(dllPath, 255);
  strcat(dllPath, "\\NOAA_32.DLL");

  gNoaaDllInst = LoadLibrary(dllPath);
}




void CallNERegister(void)
{
   char sigStr[6];
   char fullPath[256];
   char humanName[64];
   FARPROC proc=NULL;
   LoadNoaaDll();
   if((UINT) gNoaaDllInst > 32)
     {
        //we have the library
        proc = GetProcAddress(gNoaaDllInst,"NERegister");
        if(proc)
        {
           my_getindstring(humanName, 1000, 1); //ALOHA
           (*proc)(   gMySignatureStr,
                      gMyMainWindowHWND,
                      gMyMainWindowClassName,
                      "",              //messageStringForHola,unused
                      "",           // human name,unused
                      "",    //wakeUpTopicString,unused
                      "",          // fullPath,unused
                      0,           //SA_APPTASK,unused
                      0,              //unused
                      0);             //unused
      }
    }
 }
```

To send a message:

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

```
long CallNESendMessage(char* toSigStr, char* messageStr)
{

   FARPROC proc=NULL;
   long err = -1;
   if((UINT)gNoaaDllInst > 32)
   {
      //we have the library
      proc = GetProcAddress(gNoaaDllInst, "NESendMessage");
      if(proc)
      {
         err = (long)(*proc)(toSigStr, messageStr, FALSE, NULL, NULL);
      }
      return err;
   }
}
```

To receive a message:

In your WndProc(), check for messages on idle (e.g. set a timer):

```
long HandleNEMessage(void)
{
   // check for a message and handle it
   FARPROC proc=NULL;
   long err = -1;
   long len;
   long maxLength = 1023;
   char msgStr[1024]="";
   LoadNoaaDll();
   if((UINT)gNoaaDllInst > 32)
   {
      //we have the library
      proc = GetProcAddress(gNoaaDllInst, "NEGetNextMessageLength");
      if(proc)
      {
         len = (long) (*proc)(gMySignatureStr);
         if(len > 0)
         {
            //we have a message
            proc = GetProcAddress(gNoaaDllInst, "NEGetNextMessage");
            if (proc)
            {
              BOOL gotIt;
              gotIt = (BOOL)(*proc)(sigStr, messageString, maxLength);
                  if(gotIt) {
                       // code to handle the message goes here

                  }
            }
```

```
         }
      }
   }
   return err;
}
```

To see if an application is running:

```
BOOL CallNEAppIsRunning(char* toSigstr)
{
   FARPROC proc=NULL;
   BOOL isRunning = FALSE;
   LoadNoaaDll();
   if((UINT)gNoaaDllInst > 32)
   {
      //we have the library
      proc = GetProcAddress(gNoaaDllInst, "NEAppIsRunning");
      if(proc)
      {
         isRunning = (BOOL)(*proc)(toSigStr);
      }
   }
   return isRunning;
}
```

To quit:

```
void CallNEBye(void)
{
   char sigStr[6];
   FARPROC proc=NULL;
   if((UINT) gNoaaDllInst > 32)
   {
      //we have the library
      proc = GetProcAddress(gNoaaDllInst,"NEBye");
      if(proc)
      {
         (*proc)(  gMySignatureStr,
                     gMyMainWindowHWND,
                     gMyMainWindowClassName,
                  ""); // messageStringForBye, unused
      }
      FreeLibrary(gNoaaDllInst);
      gNoaaDllInst = NULL;
   }
}
```

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 necessary 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 explicitly 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 applications, 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.

IAC Dictionary 3.0 Notes

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.

=====>	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/overlay, and MARPLOT is responding. MARPLOT has written out a file containing the ID numbers of all of the objects on the map/overlay 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.

'CTL!' (continued)		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 set 1, or 256 plus the ASCII value of the character from font set 2. • 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 overlay. 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 overlay 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 you send a CTRL message with "OFF" for the MODE parameter.
	'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>" "<overlay>" "<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, <overlay> and <map> are the overlay 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 <overlay 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 <overlay name> \t <map name>
	'LSTB'	a return-delimited string where each line (one per object) has the format: <object id> \t <overlay name> \t <map name>
		Note: In the case that ACTN is DRAG, the LST lines have the format: <object id> \t <overlay 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 <overlay name> \t <map name>
		Note: If OBJ2 is empty, an error occurred during the creation of the neighborhood object, probably because P was not found.

'OBID'		Here are IDs 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 IDs, each (including the last) followed by a return; the order of the IDs 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 <overlay 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 overlay does or does not exist. If you have sent MARPLOT an OVX? message asking if a particular overlay exists. MARPLOT is responding to tell you the answer. If you have sent MARPLOT a MKOV message, MARPLOT is informing you of the overlays ID.
	'ANSR'	"Y" if the overlay exists, "N" if not
	'OVID'	ID of overlay if it exists
#	'ATBS'	attributes of the overlay 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 overlays were called overlays.
'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 <overlay 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 the entries 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> decimal strings, S and W negative, \t <long> center of object

'YROD' (continued)		# 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 you include the "TAGS" keyword, each of the items in each of the lines returned by YROD is preceded by a tag so you can identify the field using a more general parser. For instance, is you include "TAGS" and "NAME", your data lines will have the format ID: \t <object id> \t LAYER: \t <overlay name> \t MAP: \t<map name> \t NAME: \t <name> RETURN Finally, you can include the "MIE" keyword in the OPTN parameter. This keyword overrides any use of APPID, NAME, LATLONG, GRAPHICS or TAGS. It causes the objects to be exported in full MIE format, just as if the user had chose Export from the File menu in MARPLOT.
	'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 messages 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/overlay within the current view. The MODE parameter determines whether you want to start (on) or stop (off) controlling the overlay. When you start controlling an overlay, it is automatically put into “show” mode. Note that you must specify a map and an overlay. Objects on the given overlay but on a different map will appear as they would normally, as will objects on the given overlay 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 overlay
	'MODE'	"ON" or "OFF"; ON means you want to start controlling the overlay; OFF means you are done controlling the overlay. "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 overlay (an overlay 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 overlay was originally controlled (MARPLOT needs to know this view to know what the “clipping rectangle” is for the TDO file).

'CTRL' (continued)	=====>	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 overlay using the LAYR parameter. In the latter case, you can use the ' IDS ' and 'MAP ' parameters to delete only certain objects on the overlay.
	'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 <overlay 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 IDs of objects on the named overlay to be deleted
	'LAYR'	(must be given if LST2 is not given) the name of the overlay on which the objects are to be found
	'MAP '	(optional) the name of the map from which objects on the named overlay are to be deleted (objects on the overlay but on other maps will not be touched); use the name "USER" to indicate the current user map. Default is the current user map.
'DLOV'		This message is used to delete all the objects from an overlay; unless the REMOVELAYER option is used, the overlay itself is still retained in MARPLOT, in its specified position in the overlay list.
	'LAYR'	The name of the overlay
	'MAP '	(optional) The name of the map from which the named overlay is to be deleted (objects on the overlay 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 overlay on all maps. Default is 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 overlay be deleted from MARPLOT's overlay list, after the specified overlay map combination specified is deleted. **Note:** The overlay will not be deleted from the overlay list if there are objects remaining on this overlay 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; in 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 Click 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 application'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
	'FRWD'	(optional) If the first character is "N" or "n", MARPLOT will not come forward. Coming forward is the default behavior. (This optional parameter was added for MARPLOT 3.3.2)
	'MOD?'	(optional) Supports the same features as the Import Options Dialog... answering the question: "If an object being imported, Object I, has the same ID as an object already on the map, Object M:" "R" (default) = add Object I and delete Object M ("R" means Replace) "S" = do not import Object I ("S" means Skip) "A" = add Object I and do not delete Object M (this allows multiple objects with a given ID on a given map/overlay combination) ("A" means Add)

'IMP2' (continued)	'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 overlay, 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 overlay, 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 IDs 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

'IMPT' (continued)	'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 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 overlay-list legend at the time the LGND message is received, the overlay-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.

'LGND' (continued)	'XOUT'	(optional) return-delimited and return-terminated list of names of overlays NOT to be included in the legend when the user is showing a overlay-list legend at the time the LGND message is received; that is, these specific overlays are not retained in the new legend, but other overlays 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 overlays 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 — This keyword is no longer supported. NODEFAULT — This keyword is no longer supported. DELETENO (default) — objects on this overlay may not be deleted DELETEYES or DELETEOK — objects on this overlay may be freely deleted by the user DELETEALERT — when the user attempts to delete objects on this overlay, 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)

'MAP ' 'OPTN' (continued) (continued)	MOVELAYERNO (default)	the user cannot change the overlay of objects on this map
	MOVELAYERYES or MOVELAYEROK	the user can freely change the overlay of objects on this map
	MOVELAYERALERT	when the user changes the overlay 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.
	UNIVERSALNO	the map is not a universal map.
	UNIVERSALYES	the map is a universal map.
	Note: If neither flag is specified, the map is assumed to be an old style map and MARPLOT will assume the map is not a universal map (unless it is owned by 'CAMO'). A universal map is basically a map which should be considered to have the bounding rect equal to the entire world. A universal map is always in view when searching. Universal maps have no alert when the user extends the map bounds when adding an object. A universal map is not included as an entire map view in the view list. The USER MAP is always considered a universal map.	

'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 overlay. This message asks MARPLOT to create a new overlay. This will not create a new overlay if one with the same name already exists.
	'NAME'	name of overlay to create
	'ATBS'	attributes of the new overlay (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 overlay ID of the newly created overlay.
		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.

<table>
<tr><td>'MYOD'
(continued)</td><td></td><td>keyword meaning

The following three keywords are used to specify the objects of interest. These 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 this keyword has two uses. The primary use of this keyword is to indicate that the object data should be written to a file, and that the path to this file should be returned in the LST2 parameter of the YROD message. The secondary use of this keyword is that 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 overlay, search all overlays
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 are used to specify the type of information to be returned in the LST2 parameter. These 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</td></tr>
<tr><td></td><td>'PATH'</td><td>{optional; only has meaning when the FILE keyword is used}
a string containing the path to a directory where the YRODLST2.TXT file should be written. Use of this keyword overrides MARPLOT’s default location for the file, which is the friends directory;
(This parameter is new in MARPLOT 3.3)</td></tr>
<tr><td></td><td>'OPTN'</td><td>a string containing keywords from the table above</td></tr>
</table>

'MYOD' (continued)	'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 <overlay name> \t <map name> if you have specified ANYLAYER in OPTN, you can leave the <overlay name> empty and MARPLOT will search all overlays; 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 <overlay 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 constructed 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 <overlay name> \t <map name>
	'LYR2'	the name of the overlay 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)

'NBH?' (continued)	'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. Starting with MARPLOT 3.3, the created object will also be selected "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 overlay exist?
	'NAME'	name of overlay
	=====>	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 <overlay 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 <overlay name> empty and MARPLOT will search all overlays; 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 overlay, search all overlays 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

'SHOW' (continued)	'OPTN' (continued)	>>>>>> 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, deselecting 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 overlay 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 overlay 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

'SHOW' 'OPTN' (continued) (continued)	RGBCONTROL	take control of the overlay map these objects are on and hide all objects not in the list of objects in this message; The LST2 parameter has a special format; the first line is a standard LST2 line, which is used to obtain the overlay and map for all of the objects; the ID on the first line must be present, but is ignored; The subsequent lines have 28 chars each including the single return char and are of the form: <object id> <tab><fill pattern code><red value- 000-255> <green value- 000-255> <blue value- 000-255> ; Fill Pattern. Code: Use a char from '1' to 'A', corresponding to the patterns represented in MARPLOT's Fill Pattern menu. 'X' means don't override the fill pattern. 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
	UPDATE	this keyword specifies that you want MARPLOT to return information about the specified objects; this information comes in a YROD message; the resulting YROD message is the same as if you had called MYOD with the same LST2 parameter; when you specify UPDATE in OPTN, you can also specify any of the MYOD OPTN parameters to alter the contents of the information returned (IT IS NECESSARY TO SEND THE "IDS" KEYWORD WHEN USING "UPDATE")
	NONOTFOUNDALERT	this keyword specifies that you want MARPLOT to suppress the usual alert that MARPLOT gives the user when one or more of the objects cannot be found. This is useful when sending MARPLOT messages in a batch mode because the alert would stop the message processing.
	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.	

<table>
<tr><td>'SLAT'</td><td></td><td>Set the attributes of an overlay.</td></tr>
<tr><td></td><td>'NAME'</td><td>the name of the overlay</td></tr>
<tr><td></td><td>'ATBS'</td><td>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.

<table><tr><th>keyword</th><th>meaning [extra data]</th></tr><tr><td>LOCKEDYES</td><td>overlay is locked</td></tr><tr><td>LOCKEDNO or
UNLOCKED</td><td>overlay is unlocked</td></tr><tr><td>TEMPORARY</td><td>overlay is temporary</td></tr><tr><td>PERMANENT</td><td>overlay is permanent</td></tr><tr><td>APPOWNED</td><td>overlay is owned by sending app</td></tr><tr><td>MARPLOTOWNED</td><td>overlay is owned by MARPLOT</td></tr><tr><td>DELETEOK</td><td>even if overlay is owned, user can delete objects on overlay</td></tr><tr><td>DELETENO</td><td>when overlay is owned, user cannot delete objects on overlay</td></tr><tr><td># GRAPHICS</td><td>default overlay graphics [C, S, P, L, W (see YROD)]</td></tr><tr><td>INDIVIDUAL</td><td>individual graphics</td></tr><tr><td>COMMON</td><td>use default graphics</td></tr><tr><td>HIVISSCALE</td><td>scale at which overlay shows [n, where scale is 1:n]</td></tr><tr><td>LOVISSCALE</td><td>scale at which overlay hides [n, where scale is 1:n]</td></tr><tr><td>DOTSSCALE</td><td>scale at which symbols->dots [n, where scale is 1:n]</td></tr><tr><td>NAMESSCALE</td><td>scale at which names show [n, where scale is 1:n]</td></tr><tr><td>SHOWNAMESMODE</td><td>show + names mode</td></tr><tr><td>SHOWMODE</td><td>show mode</td></tr><tr><td>RANGEMODE</td><td>range mode</td></tr><tr><td>HIDEMODE</td><td>hide mode</td></tr><tr><td># NAME</td><td>name of overlay</td></tr><tr><td># POSITION</td><td>position of overlay in list ["TOP", "BOTTOM" or n (1 = top)]</td></tr></table></td></tr>
</table>

'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 within 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 overlays to be searched; an empty string indicates that all overlays should be searched. Note that unless the SEARCHSIMILARLAYERS option key is used, overlays must be explicitly listed. For example, if the parameter is "Roads ", the Roads overlay will be searched but the Roads (Major) overlay will not be searched. If you want MARPLOT to search Roads and all of the similar overlays, 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

'SRCH' 'OPTN' (continued)	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 overlays with names similar to the names in the LYRS parameter. For example, if the LYRS parameter is "Roads ", both the Roads overlay and the Roads (Major) overlay 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. NONFILLEDASFILLED — (new to MARPLOT 3.2.4) when checking polygon, circle and rectangle objects for touching or within searches, treat non-filled objects as filled objects. If this parameter is not specified, a non-filled object is assumed to represent its boundary and not its interior. (note the selection behavior of non-filled objects changed with MARPLOT 3.2.4.) The remaining keywords apply to the returned YROD 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 <overlay 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)

'SRCH' (continued)	'OPTN' (continued)	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.
'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.