NAME

mapper – GMA battle grid map

SYNOPSIS

gma mapper −− −−help

gma mapper [**−display** *name*] [**−geometry** *value*] [*other wish options*] **−−** [**−A**] [**−a**] [**−B**] [**−b** *pct*] [**−C** *file*] [**−c** [*image***=**]*name*[**:***color*]] [**−D**] [**−d**] [**−G** *n*[**+***x*[**:***y*]]] [**−g** *n*[**+***x*[**:***y*]]] [**−h** *hostname*] [**−k**] [**−l**] [**−M** *module*] [**−n**] [**−P** *pass*] [**−p** *port*] [**−s** *file*] [**−t** *transcript*] [**−u** *name*] [**−X** *proxyhost*] [**−x** *proxyurl*] [**−−button−size small|medium|large**] [**−−curl−path** *path*] [**−−curl−url−base** *url*] [**−−dark**] [**−−mkdir−path** *path*] [**−−nc−path** *path*] [**−−no−blur−all**] [**−−scp−dest** *path*] [**−−scp−path** *path*] [**−−scp−server** *hostname*] [**−−ssh−path** *path*] [**−−update−url** *url*] [*mapfiles...*]

gma mapper [**−display** *name*] [**−geometry** *value*] [*other wish options*] **−−** [**−−animate**] [**−−no−animate**] [**−−blur−all**] [**−−no−blur−all**] [**−−blur−hp** *pct*] [**−−button−size small|medium|large**] [**−−config** *file*] [**−−character** [*image***=**]*name*[**:***color*]] [**−−chat−history** *n*] [**−−curl−path** *path*] [**−−curl−url−base** *url*] [**−−dark**] [**−−debug**] [**−−generate−config** *path*] [**−−generate−style−config** *path*] [**−−guide** *n*[**+***x*[**:***y*]]] [**−−host** *hostname*] [**−−keep−tools**] [**−−major** *n*[**+***x*[**:***y*]]] [**−−mkdir−path** *path*] [**−−module** *module*] [**−−nc−path** *path*] [**−−no−chat**] [**−−password** *pass*] [**−−port** *port*] [**−−preload**] [**−−proxy−host** *proxyhost*] [**−−proxy−url** *proxyurl*] [**−−scp−dest** *path*] [**−−scp−path** *path*] [**−−scp−server** *hostname*] [**−−ssh−path** *path*] [**−−style** *file*] [**−−transcript** *filename*] [**−−update−url** *url*] [**−−username** *name*] [*mapfiles...*]

Options cannot be combined into a single CLI argument. They are evaluated in the order they are received. While not recommended (nor guaranteed to always be the case), it happens that options specified after *mapfile*s will affect only *mapfile*s named following those later options. Thus, if a *mapfile* has a name beginning with a hyphen, it is necessary to specify it in some sneaky way, such as prefixing it with a "**./**".

(Depending on your system, you may need to run as **wish mapper.tcl** ...)

DESCRIPTION

The **mapper** program displays a battle grid map for the players to see their tactical positions with respect to their opponents and features of their environment (e.g., dungeon rooms).

If one or more map file names are listed on the command line, those files are loaded into the grid display. Otherwise, a blank map grid is shown. Map files may subsequently be loaded via interactive commands or upon request from a control service (see the **−h** option and "Control Protocol" section, below).

See notes at the bottom of this manpage for installation requirements.

HELPER PROGRAMS

The **mapper** client makes use of a few other programs to facilitate the efficient handling of data transfers. These are only used if the client is working in networked mode rather than as a stand-alone map drawing program.

curl Assuming the GM (or whomever is administrating the map assets) has arranged for a web server to host content for the map, the **mapper** client will invoke the **curl**(1) program to actually retrieve the files from that server. If needed, you will need to provide options and/or configuration file parameters as documented below to let **mapper** know where to find the **curl** binary and what options it needs to find the web server, navigate through proxy servers, etc.

scp For users with privileges to *store* data on the web server (generally, this would be the GM or other designated administrative users, not the general player group), **mapper** will invoke **scp**(1) to copy new data files up to the server. For example, in store-and-forward mode, when opening a local data file or pushing map contents out to other clients, **mapper** checks to see if the server has the data file(s) available for everyone to download. If not, it will use **scp** to upload them to the server before sending out a command to peer clients to retrieve them.

This requires that the user is authorized to perform this action on the remote server. Usually this is done by having an SSH certificate loaded via **ssh-agent**(1) and the server set up to recognize that for the needed operations without requiring a password to be manually entered.

ssh Along with the use of **scp** to transfer new data to the web server, **mapper** will use **ssh**(1) to create directories and set file permissions as needed on the server for the content being uploaded.

OPTIONS

The following options control the behavior of **mapper**.

−A,**−−no−animate**

Suppress the animation effects enabled by the **−a** (**−−animate**) option. This is the default, so it only needs to be given to cancel an **−a** option which appeared previously or which was read from a configuration file.

−a,**−−animate** This option causes the mapper to make a lame attempt to "animate" the appearance of objects being drawn onto the screen by updating the display after each one. (The author was feeling oddly nostalgic about working with the Tektronix storage-screen computer displays of his childhood at the time. If you don't know what that means, go watch an episode of the 1970s-era Battlestar Galactica and look at most of their computer screens.)

−B,**−−blur−all** Normally, the imprecision introduced to health bar displays by the **−b** (**−−blur−hp**) option applies only to "monsters" (opponents). With this option, it applies to all participants.

−−no−blur−all Cancels the effect of the **−B** (**−−blur−all**) option.

−b,**−−blur−hp** *pct*

This option "blurs" the health bar displays by rounding off the displayed hit point total to only be accurate in *pct*-percent intervals. For example, a setting of **10** means the health bar will blur the value by 10%; in other words, rather than every hit point showing proportionally on the health bar, the health bar will only show 10 possible intermediate values, corresponding to the hit points being 1–9%, 10–19%, 20–29%, ..., 90–99% of the total, as well as 0% and 100%. Thus, higher *pct* values indicate less accurate displays.

Setting *pct* to 0 (or less) indicates that no blurring is desired; in this case the display is precisely accurate. This is the default, but note that the hit points reported may be blurred on the server (GM)'s side independently.

Once a creature reaches 0 hit points, no further blurring is done, so the bleed-out sequence is accurate (but this is fairly quick for almost all creatures and is of less consequence than the hit point totals while they are still alive and fighting, so this was considered a better course of action).

−−button−size *size*

Change the size of the toolbar buttons. The *size* value may be any string starting with **s**, **m**, or **l**, representing small, medium, or large-size icons. Small buttons are the default.

−C,**−−config** *file* Read a set of command-line options from the named *file* as if they appeared at this point in the list of command-line options. Only the long-form option names are allowed and are given without the leading hyphens. The file must contain a single option per line. Options which take a parameter are separated from their parameter with an equals sign (although this is currently not supported in the command line itself). For example, a configuration file might contain the following:

```
# My configuration settings
scp-dest=/usr/local/game-support
scp-server=www.example.org
curl-url-base=https://www.example.org/game
no-animate
keep-tools
```

Note that any line whose very first character is a pound sign ("**#**")) is ignored as a comment.

If the file **~/.gma/mapper/mapper.conf** exists, it is read first before command-line options or (other) configuration files are loaded.

Note that more than one **−−config** (and/or **−C**) option may be given, in which case the files are read in the order they appear in the command line. This may be used to split up options into different files, such as general settings common to all sessions, and specific settings based on networks or different games.

−c,**−−character** [*image***=**]*name*[**:***color*]
Add player *name* to the list of standard players tracked by the mapper. This is the list that appears in the pop-up menu for placing people onto the map. If *color* is also specified, that color is used for the threat zone highlighting. This may be a standard X11 color name, or an RGB value in the form **#***rgb,* **#***rrggbb,* or **#***rrrgggbbb.* The default is "blue".

If an image file will be used with the character that's not the same name as the character, specify it as *image***=***name* in this option.

Multiple **−c** options may be given. Each adds another name to the list.

Note that when the mapper is networked with the GM's console, the default list of names actually comes from the GM's console so it should not be necessary to specify these names to the client from the command line or configuration file.

−−chat−history *n* Limits the retained chat history to *n* messages. When **mapper** starts, it reloads the chat history it had cached from the previous session on the current *host* and *port* but if that results in more than *n* messages being in the history, the list of messages is truncated to the most recent *n* (both in-memory and in the cache file). Any additional messages received will be kept, so the actual history will be a little larger than *n* until the next time **mapper** is started. If *n* is less than or equal to 0, then no limit is placed on the history size. The default limit is 512.

−−curl−path *path*
Specify the path to the **curl**(1) program on your system, if the built-in default doesn't work for you. This is used when fetching image and map files from the server.

−−curl−url−base *url*
Specify the base URL on the data server. The files downloaded by mapper clients will be in a directory hierarchy appended to this string.

−D,**−−debug** Increase debugging output level. Multiple **−D** options further increase verbosity of debugging messages. These are displayed in a separate window.

−d,**−−dark** This option changes the default color palette to use a darker background which may be easier to look at for longer periods of time. On macOS systems running up-to-date versions of Tcl/Tk (not the default legacy version supplied by Apple), dark mode is automatically selected if the macOS session is also configured for dark mode.

−G,**−−major** *n* Make every *n*th gridline green (very thick lines). This is for major guide lines on the map.

−G,**−−major** *n***+***o* As above, but offset the major guide lines to the right and down by *o* lines. The **+** character is required, but the value of *o* may be negative, so the option "**−G** *n***+−3**" would move the lines to the left and up by 3 lines.

−G,**−−major** *n***+***x***:***y*
If expressed this way, rather than use the same offset in both directions, move the guide lines *x* lines to the right and *y* lines down.

−g,**−−guide** *n* Make every *n*th gridline red (thick lines). This is for minor guide lines. The value of *n* may be specified in all the same ways as for the **−G/−−major** option (see above).

−−generate−config *path*
Append a set of example settings to the file named in *path*. This will include the set of options you might want to configure in the mapper, most of which will be commented out. This is intended to make it easier for you to create a configuration file for the mapper without having to remember what all of the options are. In reality you won't likely need most of them.

After writing to this file, the mapper client will exit.

−−generate−style−config *path*
Append a set of example style settings to the file named in *path*. This will include all possible style definitions. Their built-in default values are also provided, so that if this file were loaded as the *style.conf* file of the mapper, it should result in the same appearance it normally would have. The intent is to help you get started configuring the styles of the mapper and to see what the mapper's settings would be if you didn't override them. You should delete or comment out any entries you wish to leave at the built-in default values.

After writing to this file, the mapper client will exit.

−−help Print a summary of the command invocation options and exit.

−h,**−−host** *hostname*
Connect to a map control service running on the designated host. This will send updates to item positions, display of rooms, etc. If this option is not specified, no control connection is made, and the mapper runs in stand-alone mode.

−k,**−−keep−tools** Normally, map clients have their toolbars turned off to maximize the available screen space for the battle map. The GM can turn on and off their toolbars from his console as needed. If this option is given, this causes the client to unconditionally display its toolbar anyway. This is used for the main map run by the GM or whomever else is managing the group map and needs the toolbar active, or if people just want to keep the toolbar all the time.

−l,**−−preload** Pre-load all the cached images into memory at start-up, instead of loading them as needed during the map's operation. Note that this only loads cached images which are new enough that the mapper wouldn't check the server for newer versions anyway, thus allowing a mapper client to be restarted mid-game with a minimum of impact to game performance.

−−mkdir−path *path*
Specify the *server-side* path to the **mkdir**(1) program which will be used when uploading files *to* the data server (authorized users only).

−M,**−−module** *module*
Use the module ID code *module* for this session. This is used to differentiate server-side resources between campaigns which have conflicting names. This is only needed for the mapper clients used as the forwarder in store-and-forward mode (typically the GM's own client).

−−nc−path *path* Specify the path to the **nc**(1) program which will be used when sending files *to* the data server (authorized users only) through a SOCKS proxy server.

−n,**−−no−chat** Suppress the display of incoming chat messages including die rolls.

−P,**−−password** *pass*
For servers which require authentication, this specifies the password to gain entry to that server. This is a fairly simple authentication mechanism intended to block nuisance connections, spam, and accidental connections of legitimate clients to the wrong game server. If *pass* is given as a single question mark ("**?**"), then the user will be prompted to enter their password manually when connecting to the server. This avoids placing the plaintext password on the command line or in a configuration file.

−p,**−−port** *port* If the **−h** (**−−host**) option is given, connect to the specified TCP *port* number on that host. The default is port 2323.

−−scp−dest *path* Specify the *server-side* directory into which files will be uploaded (authorized users only). This will be the top-level data directory for the mapper; subdirectory names will be appended to this string.

−−scp−path *path* Specify the path to the **scp**(1) program which will be used to send files *to* the data server. (Authorized users only.)

−−scp−server *hostname*
The host name of the storage server. Only used when sending files *to* the server (authorized users only).

−−ssh−path *path* Specify the path to the **ssh**(1) program used to send files *to* the storage server (authorized users only).

−−style *file* Loads custom style definitions for fonts, colors, and so forth from the named *file*. If this option is not specified but the file **~/.gma/mapper/style.conf** exists, that file will be read by default. See **style.conf**(5) for details about what can go in this file.

−t,**−−transcript** *path*
Records all chat window activity (including results of die rolls) to the specified file *path*. If this file exists, it will be appended to with the new information.

The following special tokens may appear in the *path* string, which will be replaced with values based on the time of day the file is opened:

%a Mon, Tue, etc.

%A
Monday, Tuesday, etc.

%b
Jan, Feb, etc.

%B
January, February, etc.

%d
Day of month (1−31).

%j Julian day of the year.

%m
Month (01−12).

%y Year (00−99).

%Y
Year (all digits).

%H
Hour (00−23).

%I Hour (01−12).

%M
Minutes (00−59).

%S
Seconds (00−59).

%p
AM or PM.

%D
Date (%m/%y/%d).

%r Time (%I:%M:%S %p).

%R
Time (%H:%M).

%T
Time (%H:%M:%S).

%Z
Time zone name.

−−update−url *url* Specifies the URL where updated versions of the **mapper** program may be obtained. This enables automatic upgrades. The **mapper** will, with the user's approval, download updated versions of itself from this URL and install them.

−u,**−−username** *name*
Sets the name used to identify you amongst the other players on your server. If this option is not provided, your current system username will be used instead.

−X,**−−proxy−host** *host*[**:***port*]
For sending files *to* the server (for authorized users only), use the specified SOCKS5 proxy server. (E.g., **−X proxy.example.org:1080**.)

−x,**−−proxy−url** *proxyurl*
Use the given URL to connect through a proxy server to fetch image data. This does not affect the connection to the map server used by GMA (yet). (E.g., **−x http://proxy.example.org:1080**.)

INVOCATION

As of this writing, the mapper still has not been ported to the new GMA code in Python, and is still implemented as a Tcl/Tk script. This means you need to have a Tcl/Tk interpreter installed on your system. (See **http://tcl.tk** if you need more information about that.) Since it's a GUI application, it is run using the **wish** command (the Tcl Windowing Shell).

We have noted that at least on the Mac platform, the **wish** program will refuse to let you expand the map window larger than the largest dimensions of the screen(s) when it was launched, so you want to plug in any projector or external displays before starting the map.

INTERACTIVE USAGE

The mapper shows the dungeon area around the players and includes features which are helpful for managing game mechanics, particularly those relating to combat. It is intended to be fairly self-explanatory (and I don't have time to thoroughly document everything at the moment), so the following brief notes will hopefully suffice to help a new user navigate its quirks.

The system menu bar is not used for this application, and is left to whatever the **wish** program sets it to for generic scripts. Instead, all of the interaction with the mapper is done through its toolbar and context menu.

Tool Bar

Across the top of the map is a graphical toolbar. Click on each button to activate its features. Note that some of these turn on/off different modes of operation for the map. When this happens, the mouse cursor will change to show the mode the map is currently in.

Each button is described briefly below. The first block of buttons control the mapper's mode of operation. They function as radio buttons (only one is active at a time, and selecting one de-selects the previously selected one).

Line Tool (cross-hair cursor)
Selects the line drawing tool. When this tool is active, click the left button to start drawing a line, then click it again at the other end of the line. You may continue clicking to get multiple connected line segments (which all count as a single object on the map). When finished, press

the Escape key or click the middle button. Cancel by pressing Escape or the middle button without having defined any points on the line at all. Note that the current FILL color (not OUTLINE color) is used to draw the line on the map.

Rectangle Tool (square cursor)
: Selects the rectangle drawing tool. When this tool is active, click the left button where you want one corner of the rectangle to go, then click again where the diagonally opposite corner should go. The rectangle will be outlined in the OUTLINE color and filled in with the FILL color. Cancel by pressing Escape or the middle button.

Polygon Tool (polygon cursor)
: This works like the Line Tool except that the region inside the shape defined by the line segments is filled in with the FILL color, while the outline is colored in the OUTLINE color.

  Note that when this tool is selected, the two option buttons become active, offering some different options for how the lines of the polygon are to be joined:

  Corner
  : Each time you click on this button, it cycles through the different corner-join options: beveled, mitered, and round.

  Spline
  : Each time you click on this button, it cycles through the spline levels from 0 (no splines, just straight lines), to 9 (use 9 lines between points to make a smooth curve).

Ellipse Tool (circle cursor)
: This works like the Rectangle Tool except that it draws an elliptical shape inscribed within (tangent to) the rectangular area defined by the two mouse clicks.

Arc Tool (diamond crosshair cursor)
: This tool is for making various semicircular shapes. Its operation is a little more complex than the others. When this tool is active, the option is also active, allowing you to choose the type of arc to create:

  Arc type
  : Each time you click on this option button, it cycles through the choices of arc types: pie slice, chord, and arc.

  First, draw the elliptical shape for the arc (as if it were a complete ellipse) as described for the Arc Tool. Then, move the mouse horizontally to rotate the arc and vertically to adjust the length of the arc. When satisfied, click the left button to complete the arc. Cancel by pressing Escape.

Text Tool (i-beam cursor)
: This is used for placing text on the map. Its operation works much like the stamp tool (q.v.), in that left-clicking on the canvas will place a new copy of the current string at that location. If there is no current string, you will be prompted to enter one. Right-clicking will prompt you for a new string rather than using the current one. The current string is displayed below the tool bar.

  With this tool active, a font selection button is available. Clicking this toggles the font selection dialog. Changing the font in that dialog will alter the font of the most recently placed text object (as long as the text tool remains active) and sets the font for future text objects.

  There is also an anchor selection button while this tool is active. This shows as a centered cross (+) to indicate that the text will be centered around the point where the mouse is clicked. Clicking on the anchor selection button will cycle through all of the anchor directions available: north, south, northeast, etc. These mean that the text will be aligned so that the point where the mouse is clicked will be that direction from the text. Thus, for example, selecting an anchor of "west" (incidated by a left-pointing arrow) will center the text vertically but align it horizontally so that the point is to the left of the text.

Move Tool (iron cross cursor)
: This is the default mode, and the one you should keep the mapper in when not changing the map features. With this mode, you can drag creatures around the map as described below.

If the mouse is not over a creature token when starting to drag the mouse, the map grid itself is dragged, providing an easy way to scroll the map.

If you hold down the shift key while clicking the left button on the canvas in this mode, it will briefly show a marker to draw attention to the grid square the mouse is in.

Cut Tool (skull cursor)
: With this tool active, any object you click on with the left button will be deleted from the map immediately (no saving throw). if you click where there are multiple overlapping objects, you will be prompted to select which to delete. Press Escape if you don't want to delete any of them.

Object Move Tool (multi-arrow cursor)
: This tool allows the map objects (as opposed to creature tokens) to be dragged to new locations. Note that you are dragging the object's *reference point* with the cursor. Once an object has been moved any distance with the mouse, the arrow keys (or the standard **vi**(1) movement keys) may be used to "nudge" the object by one pixel at a time up, down, left, or right; additionally the keys **u**, **d**, **f**, and **b** may be used to move the object up, down, to the front, and to the back in the stacking order (z coordinate), respectively.

Stamp Tool (star cursor)
: This allows graphical tiles to be "stamped" onto the map. If there is a current tile already chosen, a new copy of it is placed on the map with the upper-left corner at the point the mouse was clicked. If no such tile was chosen, you will be prompted for its name. Right-clicking will force the selection of a new tile image rather than re-stamping the current one. See **rendersizes**(6) for more information about the format of these tile files. They should be rendered and (if using a map server) uploaded ahead of time so they are visible in the map.

The next block of buttons control the appearance of any new objects added to the map.

Fill Mode
: Clicking on this button toggles whether the shape will be filled or not. (Somewhat counter-intuitively, lines are filled with the FILL color, not the OUTLINE color, so turning off fill will just give you invisible lines.)

Fill Color
: Clicking on this button selects the FILL color to be used to fill in new object areas. This is disabled if fill mode is turned off.

Outline Color
: Clicking on this button selects the OUTLINE color to be used to draw around new object areas.

Grid Snap
: Clicking on this button cycles through the grid snap options:

  Off — Points may be added anywhere on the canvas (free form drawing).

  1 — Points may only be added at the intersections of grid lines.

  1/2 — Points may be added at grid intersections, and 1/2 way between them horizontally or vertically.

  1/3 — Points may be added at grid intersections, and every 1/3 of the way between them horizontally or vertically.

  1/4 — Points may be added at grid intersections, and every 1/4 of the way between them horizontally or vertically.

Line Width
: Clicking on this button cycles through the line widths from thinnest to thickest.

The next block of buttons clear the map:

Clear Features
: Clicking this button wipes the map clean except for creatures.

Clear Creatures
: Clicking this button removes all creatures from the map.

The next block gives access to tactical displays.

Toggle Combat Mode
: Normally, the GM console will automatically turn on combat mode, but if you want to manually enable or disable it, click this button. When active, the threat zones around each creature are highlighted using colored cross-hatch patterns.

  If health tracking is in effect (i.e., for creature objects which have a non-empty **HEALTH** attribute), a health bar is displayed across the bottom of each creature's token. The appearance of this bar depends on the current health of the creature. For the description that follows, the significant health statistics are:

  t The total number of hit points the creature has when at maximum health.

  x The extra points (below zero) which define the amount of lethal damage a dying creature can sustain before being dead. In Pathfinder and compatible d20 games (and perhaps others), this is the Constitution score for the creature.

  l The number of hit points worth of *lethal* damage sustained by the creature.

  n The number of hit points worth of *non-lethal* (i.e., subdual) damage sustained by the creature.

  The health bar indicates graphically the creature's health condition and relative amount of damage they have taken, as follows:

  Full health A creature in full health will have a solid green bar across the entire width of their token's space on the map.

  Injured The full width of the token space represents the creature's total (maximum) hit points (t). A red bar will start encroaching over the green in proportion to the number of lethal hit points (l) they have taken. A yellow bar will likewise represent the number of non-lethal hit points (n) taken. Thus, the health bar will be shifting more from green to red/yellow as the creature gets more and more injured, until as it nears the point of meeting its maker, the entire bar will be red.

  Flat-footed A flat-footed creature (which does not also have any of the conditions listed below) will have a blue frame around the health bar.

  Staggered When staggered due to non-lethal damage (i.e., $n>0$ and $l+n=t$), the health bar has a yellow frame around it. The creature will move to unconscious if it suffers more damage.

  Unconscious When unconscious due to non-lethal damage (i.e., $n>0$ and $l+n>t$), the health bar has a violet frame around it.

  Disabled When disabled, a red frame will appear around the health bar. The mapper will automatically assume disabled condition if a creature has exactly 0 hit points left (i.e., $l=t$.)

  Dying When at negative hit points but still above the death level ($-x<t-l<0$), a red frame will appear but the red bar will retreat to the left as more lethal damage is taken, until it's fully black at the point of death.

  Stable If dying but stabilized, the health bar will have a brown frame around it.

  Dead When completely mortally wounded ($t-l<=-x$), the health bar is solid black.

Show HP Values

This toggles the display of health statistics for players (not monsters) over the health bars. If only lethal damage has been inflicted, it displays "*hp*/*max*" where *hp* is the current number of hit points remaining, out of a maximum of *max* hit points. If non-lethal damage has been suffered, then the display is "*hp*(*nl*)" where *nl* is the amount of non-lethal damage. If a creature is fully dead, it simply says "**DEAD**".

Spell Area of Effect

This adds a spell area of effect to the battle grid. Once created, this becomes a permanent map feature which may be removed using the Cut Tool (q.v.). When this tool is activated, two option buttons are enabled which allow you to control the shape of the spell area:

Shape This button cycles through the supported spell shapes: radius, cone, and ray.

Spread This button toggles whether the spell effect "spreads" around corners. This is not yet implemented.

Select the point of origin for the spell by clicking the left button over a grid intersection (the tool will snap to intersections automatically). Then move the mouse to the target point of the spell and click again to complete the area. As you move the mouse, the spell's area and affected grids will be shown. The area of effect is filled in with cross-hatch patterns in the FILL color. Cancel by pressing Escape. This is an active tool like the other drawing tools. When finished, select another mode such as the Move Tool.

Ruler Tool

Selecting this tool allows you to measure the distance along a path. Click the left button on a point, then move the mouse to another point. If desired, multiple points may be clicked to build a path. Middle-click or press Escape to end the measurement.

Grid Display Toggle

Clicking this button turns on and off the display of the gridlines on the map. This is a local display setting only, and is not broadcast to other clients.

Die Roller

If connected to a server, this button brings up the chat window. In this window, you may send and receive messages and die rolls to other connected users. This window is split into three adjustable panes, described individually below. The division between each pane may be moved by dragging the mouse over the separation point or pane handle.

Chat Messages Pane

In the main portion of this pane displays incoming chat messages. Each is prefixed with the name of the sender. If the message was addressed only to you, the tag "*(private)*" is added after the sender's name. If it was sent to a specific subset of users, their names will be listed as "*(private to alice, bob, charlie)*".

There are two entry lines below the chat window. The top one is for sending chat messages. Anything typed in the entry box will be transmitted when the Return key is hit. To the left of this entry box is a menu button which controls who the message is sent to. If "(all)" is selected, the message is sent to all listening clients (which need not be listed in the menu; the message will be sent to everyone at the server level). If a recipient's name is selected, it will only be sent to them. If another recipient's name is selected, they are *added to* the list of recipients. These selections are actually toggles—selecting a recipient's name again will remove them from the list. This allows for messages to be sent to any arbitrary subset of users. Selecting "(all)" will clear all selections again. The "refresh" button to the right of the entry box will update the recipient selection menu with the current set of logged-in users.

The bottom entry line is for making die rolls. Into the entry box you may type any die roll string such as would be accepted to the **DieRoller.do_roll()** method as documented in **dice**(3). When the Return key is pressed, this die roll is sent to the server, which will roll

the dice and transmit the results just like a chat message (which includes the currently-selected chat recipient list). The "(i)" button to the right of the entry box will bring up a help window explaining what may be entered for die rolls.

Recent Rolls Pane

The most recent 10 rolls entered into the above-mentioned entry box are kept in a list in this pane, with the most recent on top. Clicking on the die button next to any of these will re-roll it again. If additional modifiers are in play, they can be typed into the entry box next to the die button. Whatever is entered is simply appended to the original die expression after a plus sign. Thus, entering "5" will add "+5" to the roll, and entering "1d6 fire+3" will add "+1d6 fire+3" to the roll.

Preset Rolls Pane

A set of commonly-needed die rolls may be pre-set into the tool and then invoked using the third pane. Clicking the "(+)" button will add a new preset by prompting for its name, description, and die roll. The name uniquely identifies the preset within the list. The description will appear as a tooltip for your reference when looking at your presets. The new preset is saved on the server and will be loaded into your client every time it's started. Presets are invoked in the same manner as described above for recent rolls. Clicking the "(−)" button removes the preset from the list.

If a preset name includes a vertical bar (e.g., "**12|WillSave**"), then only the part after the bar will be displayed on-screen, but the entire name is used to sort the presets in the window. This allows arbitrary sort ordering without cluttering the display.

The file load and save buttons at the bottom of the pane are used to load and save the preset list to local disk files which have the format documented in **dice**(5).

The final block of tool buttons control global operations of the mapper:

Zoom In
: Double the visual size of grid blocks.

Zoom Out
: Halve the visual size of grid blocks.

Un-Zoom
: Restore the zoom level to normal.

Load
: Add all the map objects from a disk file onto the map tool, replacing all the map features previously on the map (but not the creatures).

Merge
: Like Load, but add to the existing objects rather than replacing them.

Unload
: All the objects saved to a selected disk file are *erased* from the map.

Push
: Push the entire contents of this map client to all other clients, replacing their current contents. (Only available in store-and-forward mode, generally only for GM use. Since the server now tracks game state and clients and re-sync with it directly, there is no longer a need for clients to push their contents to each other, and that was a problematic operation anyway.)

Store and Forward
: Toggles store-and-forward mode. When enabled, this changes the behavior of the following other buttons, providing a client update path that is much more efficient and less error-prone than streaming the object updates through the server. Stand-alone (non-networked) map clients should use the normal mode of operation instead.

  Load
  : Prompts for the selection of a map file from disk as usual. However, rather than loading that file directly, it checks to see that the file is available from the server by checking the local cache and (if necessary) downloading from the server. If the file is not found by those operations, it will be uploaded to the server (assuming the user has the proper SSH access active at the time). Other clients are then instructed to load the map file from the server.

Merge As with the Load button, but merges the map file with the existing map contents rather than replacing them.

Unload Ensures that a server copy of the map file exists as the Load button does, but then instructs the remote clients to delete the contents of that file rather than sending individual object deletion commands over the network to them.

Push Saves the current map contents to a temporary file, uploads it to the server, and then instructs the other clients to load that file.

Sync *(Note that this button's function has changed as of version 3.25.)* This clears the contents of the mapper client and requests a fresh set of data from the server, thus synchronizing this client to be in line with the server's idea of the current game state. Depending on how the server is configured, it may automatically perform this operation for you when you connect to it.

Save Save everything on the map to a disk file. You will be prompted to decide whether this includes creatures as well.

Exit Exit the mapper program.

Context Menu

Clicking the right button over an object calls up a context-sensitive menu with the following options. Not all options will be enabled in all cases. Most of these involve performing actions on creatures.

Remove *name* Remove the creature from the map. If there are multiple creatures in the same grid, a submenu will allow you to select which one to remove, or allow you to remove them all.

Add Player... Add one or more new player tokens into the grid clicked. This pops up a dialog box to enter the relevant information about the new player:

name The name by which the creature is to be known on the map. This *must* match the name the GM console is using to track initiative, or it'll never be highlighted when its turn comes up (otherwise the name doesn't matter). If the name coincides with graphical tile images already loaded, that image will be used instead of a plain circle with the creature's name inside. If a range in the form **#***n*−*m* is appended to the name (usually with a space between the name and this notation), then *m*−*n*+1 copies of the creature are added in a series of grid spaces starting with the one right-clicked and continuing to the right. For example, entering the name "Orc #1−3" will create three creatures, named "Orc #1", "Orc #2", and "Orc #3". Names must be unique. If another token was already on the map with the same name, it is replaced with this one.

If a different image file is needed than the default (named the same as the person's name), specify it as *imagename*=*creaturename* (optionally followed by the **#** notation described above).

size The size, in units of grid squares (diameter), of the creature's token. You can also use standard size designations **f** (fine), **d** (diminutive), **t** (tiny), **s** (small), **m** (medium), **l** (large), **h** (huge), **g** (gargantuan), **c** (colossal). Where it makes a difference, indicate "tall" creatures by using a capital letter and "wide" creatures with a lower-case one. Since the recommended practice is to use the size codes, which means you would use the same code for both *size* and *area* fields, any time you type into the *size* field, that will update *area* at the same time. If a different *area* is needed, that can be edited afterward separately.

area The threat area in the same units as the *size* field. This may also be one of the standard size designator codes as with *size* (and this is generally preferred). In that case, for size categories larger than medium, use upper-case (tall) letters for size categories of tall creatures, and lower-case for

long creatures.

color The color of the threat zone to draw around the creature in combat mode.

reach? Check this box if the creature has a reach weapon in hand.

Clicking **Ok** places the creature(s) on the grid and dismisses the dialog box, while clicking **Apply** places the creature(s) but leaves the dialog up in case you want to add more creatures to that grid square.

Add Monster... Just like **Add Player** but adds a monster token.

Toggle Death for *name*

Flips the creature token between living and dead states. The mapper will automatically draw an "X" across the creature token in addition to switching to the "dead" image (if images are used).

Cycle Reach for *name* Cycles through the extra threat zone for reach weapons. This goes from normal threat area to reach area and then to extended reach (both adjacent and reach zones together), then back to normal again.

Toggle Spell Area for *name*

Defines a spell effect which is described as a radius "centered on you" (or some creature). After choosing this item, click the left button to define where the radius extends from the creature's perimeter. If there was already a spell in effect, this cancels it. This differs from the spell area tool from the toolbar in that it moves with the creature and radiates from the creature's entire space rather than coming from a fixed point on the map. The area is filled in with the current FILL color.

Polymorph *name* If alternative images are available for a creature, this selects which is to be displayed. If the creature has a **SKINSIZE** attribute which indicates the size of each of its polymorphed forms, then this menu will allow you to choose between the number of forms defined for that creature, and will automatically adjust the creature size at the same time. Otherwise, the mapper program doesn't know what alternate forms are available so it will offer you a choice of three different forms, and will make a best-effort attempt to locate and display the corresponding images. In this case, you will need to manually adjust the size if needed.

Change Size of *name* Alter the size of a creature token.

Toggle Condition for *name*

Selects a condition from the list of conditions built in to **mapper** or defined by the map service for custom game-specific conditions. If the selected condition is already set for the target creature(s), then it is removed. Otherwise it is added to the target(s).

Tag *name* Add a tag to a creature token to indicate their conditions. The recent tags which were set are remembered and available in a sub-menu for convenience.

Set Elevation of *name* Specify how high above (or below) the obvious reference level a particular creature is. This puts a tag in the upper right corner of their token in which is shown their elevation (as a simple number). The sub-menu triggered by this item allows easy selection of relative distances, so you can quickly note that a creature moved up by 10 feet, for example. Any arbitrary elevation may be directly input by selecting "**(set)**" and typing the desired elevation into the dialog box that appears. If an absolute number is input, that will be the new elevation. If the number begins with a **+** or **−** sign, its value will be added or subtracted to the current elevation instead.

Set Movement Mode Various modes of locomotion are denoted in the elevation tag (q.v.) by using a different color for each. Use this menu item to select the mode currently employed by the creature:

land (white text on a black background)

fly (black text on a deep sky blue background)

climb (white text on a forest green background)

swim (white text on a teal background)

burrow (white text on a sienna background)

Deselect All Cancels the multiple-creature selection.

Show Visible Objects Moves the scrollbars to bring map features into view.

Sync Others Views Moves all the other map clients scrollbars to see what this client is showing.

Refresh Display Redraws the contents of the local mapper client. This does not reload any data (see the **Sync** button in the toolbar for that), but just locally re-draws everything again. This is useful, for example, if the client didn't know about image data for tiles or creature tokens when it first rendered the display. Often, it will work in the background to discover the missing image data, so refreshing the display will then render everything properly.

name Add a player token for the named player to the map, or move it to this location if it was already on the map.

Creatures

With the Move Tool (q.v.) selected, click and drag creatures to move them around the map.

If you hold the control key down while left-clicking on creature tokens, it toggles whether that creature is included in the group selection. When a group is selected, dragging any member of the group moves the entire group at once. Context selections will also apply to the entire group (e.g., to toggle death for all the selected tokens).

In combat mode, the area threatened by each creature is shown as a dashed outline, and is cross-hatched when that player's turn is up for action. Arrows are drawn between creatures in range to be melee targets.

CONTROL PROTOCOL

When connected to a control service (see the **−h** option above), the mapper and remote service exchange commands to indicate changes to the map display. To help keep communications simple and to help clients recover from malformed or missing data, each command is a newline-terminated line of text as described in detail below. If the mapper sends one of these commands to the service, it is indicating that the local user made those changes and requests that other connected map clients update themselves accordingly. If the mapper receives the command from the service, it should comply to make the corresponding change to itself.

The software **MUST** allow Unicode text encoded as UTF-8 everywhere (of which 7-bit ASCII is a subset).

Each command (sent or received) consists of a command word followed by a JSON object holding the parameters appropriate to that command. (See the description of the **//** command for an exception to this rule.)

The JSON object may be omitted (particularly if the command in question has no parameters). Any expected parameter fields which do not appear in the JSON object are automatically assumed to have an appropriate "zero" value (i.e., empty string, zero numeric value, empty list, boolean false, etc.). Any fields which are not expected are silently ignored.

JSON field names are matched case-sensitively. A client **MAY** match names regardless of case but **MUST** emit them as documented here.

Field values are strings unless otherwise noted.

Unless otherwise noted, there is no response expected from these commands. Even where a response is expected, it will be asynchronously sent. In any case the client **SHOULD NOT** wait for a server response before considering its request to be completed. As a special case, a client **MAY** be implemented to wait synchronously for a server response during the authentication negotiation stage.

This describes protocol version 402. These notes are intended to be detailed enough to implement a client from and should be considered the definitive standard reference to the protocol.

In the protocol command descriptions, **BOLD** text indicates literal text to be sent as-is, *Italics* indicate a parameter whose value should appear in place of the name shown, and [square brackets] surround optional items which may be omitted.

The normal course of actions at the start of the conversation is for the server to immediately send an initial greeting beginning with a **PROTOCOL** command followed by comments, **AC**, **DSM**, **UPDATES**, and/or **WORLD** commands, then send an **OK** command to indicate to the client its protocol version and to issue an authentication challenge (if configured to authenticate). It is **RECOMMENDED** that the initial greeting be limited only to comments, waiting until after authentication to send anything more specific about your game.

After the **OK** command, the client responds with an **AUTH** command if it is required to authenticate. The server will respond with **GRANTED** or **DENIED** and then **MAY** issue more of the initial greeting commands listed above.

Finally, the server will issue a **READY** command to the client and begin normal interaction with the client.

The server **MUST NOT** send the **AC**, **DENIED**, **GRANTED**, **OK**, **READY**, **UPDATES**, or **WORLD** commands after the initial **READY** is sent. Clients MUST NOT send the **AUTH** command after receiving the server's **READY** signal.

Command Summary

The messages sent by clients and servers is summarized in the following table and then explained in detail in the following paragraphs. In the table, Message is the standard message name recommended for identifier names in source code, Command is the string actually sent over the network, C↔S indicates whether the message is only sent from client to server, only from server to client, or if it could be sent in either direction. Priv indicates if the command may only be sent by a client authenticated as the GM. Note that Reply indicates the command which **SHOULD** (eventually) arrive in response, but no guarantee is made as to if or when that happens since all communications are asynchronous.

Message	Command	C↔S	Reply	Priv	Payload	Description
Accept	`ACCEPT`	C→S	—	no	JSON	Subscribe to a set of messages
AddCharacter	`AC`	C←S	—	N/A	JSON	Add primary PC to party
AddDicePresets	`DD+`	C→S	`DD=`	no	JSON	Add presets to user's set
AddImage	`AI`	C↔S	—	no	JSON	Add image to client's known set
AddObjAttributes	`OA+`	C↔S	—	no	JSON	Add values to an attribute
AdjustView	`AV`	C↔S	—	no	JSON	Scroll clients' views
Allow	`ALLOW`	C→S	—	no	JSON	Indicate supported features
Auth	`AUTH`	C→S	`GRANTED`	no	JSON	Authenticate to server
Challenge	`OK`	C←S	`AUTH`	N/A	JSON	Ask client to authenticate
ChatMessage	`TO`	C↔S	—	no	JSON	Send chat message
Clear	`CLR`	C↔S	—	no	JSON	Remove objects from client
ClearChat	`CC`	C↔S	—	no	JSON	Clear the chat history
ClearFrom	`CLR@`	C↔S	—	no	JSON	Remove elements from a map file
CombatMode	`CO`	C↔S	—	yes	JSON	Turn on/off combat mode
Comment	`//`	C←S	—	no	any	Human-readable comment
DefineDicePresets	`DD`	C→S	`DD=`	no	JSON	Store new presets for user
Denied	`DENIED`	C←S	—	N/A	JSON	Server denies access
FilterDicePresets	`DD/`	C→S	`DD=`	no	JSON	Remove some presets for user
Granted	`GRANTED`	C←S	—	N/A	JSON	Server grants access
LoadArcObject	`LS-ARC`	C↔S	—	no	JSON	Add an arc to the map
LoadCircleObject	`LS-CIRC`	C↔S	—	no	JSON	Add an ellipse to the map
LoadFrom	`L`	C↔S	—	no	JSON	Load elements from file
LoadLineObject	`LS-LINE`	C↔S	—	no	JSON	Add a line to the map
LoadPolygonObject	`LS-POLY`	C↔S	—	no	JSON	Add a polygon to the map

LoadRectangleObject	LS-RECT	C↔S	—	no	JSON	Add a rectangle to the map
LoadSpellArea-OfEffectObject	LS-SAOE	C↔S	—	no	JSON	Add an area of effect to the map
LoadTextObject	LS-TEXT	C↔S	—	no	JSON	Add some text to the map
LoadTileObject	LS-TILE	C↔S	—	no	JSON	Add a graphic tile to the map
Marco	MARCO	C←S	POLO	N/A	none	Check if client is alive
Mark	MARK	C↔S	—	no	JSON	Visually mark a spot
PlaceSomeone	PS	C↔S	—	no	JSON	Place creature token
Polo	POLO	C→S	—	no	none	Acknowledge server's ping
Priv	PRIV	C←S	—	N/A	JSON	Priveleged command denied
Protocol	PROTOCOL	C←S	—	N/A	int	Signal protocol version in use
QueryDicePresets	DR	C→S	DD=	no	none	Request user's presets
QueryImage	AI?	C↔S	AI	no	JSON	Ask for definition of image
QueryPeers	/CONN	C→S	CONN	no	none	Request list of peer clients
Ready	READY	C←S	—	N/A	none	Server sign-on complete
RemoveObjAttributes	OA-	C↔S	—	no	JSON	Remove values from an attribute
RollDice	D	C→S	ROLL	no	JSON	Initiate a die roll
RollResult	ROLL	C←S	—	N/A	JSON	Result of a die roll
Sync	SYNC	C→S	any CC	no	none	Request replay of commands
SyncChat	SYNC-CHAT	C→S	ROLL TO	no	JSON	Request replay of messages
Toolbar	TB	C↔S	—	yes	JSON	Turn on/off client toolbar
UpdateClock	CS	C↔S	—	yes	JSON	Change the current time of day
UpdateDicePresets	DD=	C←S	—	N/A	JSON	Receive user's presets
UpdateInitiative	IL	C↔S	—	yes	JSON	Update the initiative list
UpdateObjAttributes	OA	C↔S	—	no	JSON	Update object attributes
UpdatePeerList	CONN	C←S	—	N/A	JSON	Notify of all connected peers
UpdateProgress	PROGRESS	C↔S	—	no	JSON	Indicate progress on task
UpdateStatusMarker	DSM	C↔S	—	yes	JSON	Define creature status marker
UpdateTurn	I	C↔S	—	yes	JSON	Indicate turn in combat
UpdateVersions	UPDATES	C←S	—	N/A	JSON	Advertise software updates
World	WORLD	C←S	—	N/A	JSON	Campaign world info

Command Details

The client/server messages and their parameters are detailed below, ordered by the server command string sent or received.

// This is a server comment. The entire command should be ignored by all clients. This is used to inject informative context and messages into the client/server conversation which may be of interest for debugging or interactive use.

This is one of two exceptions to the rule that a command contains a JSON parameter object. In this case, the entire *line should be ignored by the client and not interpreted further.* This allows, for example, comments to be sent at the start of the server's signon message to provide a human-readable declaration as to the allowed usage of the server.

PROTOCOL *v*

This command, which, if used at all, MUST be the first command sent by the server to the client, gives an up-front indication to the client as to what protocol version is expected by the server. Older versions of the server waited until the **OK** command to notify the client, but as of protocol version 400 this is now too late for the client to correctly process the commands that may be sent before the **OK**. This, along with the **//** server comment command, are the only two commands which do *not* have a JSON data payload attached to them. The protocol version number is simply sent as an ASCII string of digits separated from the **PROTOCOL** command word by a space.

Servers which implement protocol versions 400 and later MUST send this message at the start of their conversations. Servers which implement older protocol versions SHOULD send it.

AC Add a character to the pop-up menu for map clients. This makes it easy to place important character tokens on the map and ensures that such tokens are given consistent *id* values rather than generating a random one. Clients MUST NOT send this command to each other; it is intended for the server to send to the clients. The payload includes the same fields as the **PS** command.

ACCEPT A client sends this message to the server to indicate that from this point forward (until another **ACCEPT** command), only the commands listed should be sent to it by the server. The payload is a JSON object with this parameter:

Messages *(list)*
A list of command names the client wishes to receive. This is a list of command names as they appear in this specification (e.g., **["AI","CO","TO"]**). If it contains the special value "*****" or is empty, this means to accept all messages. Note that the server may still decide to send messages such as comments or the **MARCO** command (or others) at its discretion despite this request from the client.

AI Add an image to the map. (Supersedes the function of the **AI**, **AI:**, **AI.**, and **AI@** commands as of protocol version 400.) The payload is a JSON object with the following fields:

Name The name of the image as known within the mapper.

Sizes *(list of objects)*
A list of sizes available for this object. This will be *merged* with any previously defined sizes for the same image name. Each element of this list is an object with the following fields:

File The filename or server ID by which the image can be retrieved.

ImageData *(base64-encoded bytes)*
If non-empty or not **null**, this provides the raw image data. This is DEPRECATED but still supported. Instead, images SHOULD be loaded to the game server and their ServerID used by clients.

IsLocalFile *(bool)*
If **true**, then *file* gives a local file pathname for the image; otherwise it gives the server-specific ID which the client will use to retrieve the file from the server.

Zoom *(float)*
The magnification level this bitmap represents for the given image. Typically images are provided for *zoom* levels of 0.25, 0.5, 1, 2, and 4.

This does not *draw* the image on the map; it merely defines it so the client knows what to draw when an image of the given *name* at the specified *zoom* factor is called for.

Note on clients retrieving images by server ID when *IsLocalFile* is **false** and *File* contains a server-side ID: as currently implemented, the **mapper** client checks to see if it has a cached version of the file. If so, and that file is newer than 2 days, it is used without further checks. If the cached file is older, the server is queried to see if it has a newer version of the file; if so, that is retrieved and cached; otherwise, the existing cache is updated to note that it was the known latest version as of that moment. If no usable cache file is available, the file with the given id is obtained from the server. For example, if the image id were "**abcdefg**", the URL of the file to be retrieved would be *base***/a/ab/abcdefg.gif** where *base* is the base URL as specified to the **−−curl−url−base=***base* option (or **curl−url−base=***base* configuration file line).

Servers SHOULD be set up to provide alternative file formats so that, for example, with ID

abcdefg files such as **a/ab/abcdefg.jpg** or **a/ab/abcdefg.png** may be retrieved for clients which use those other formats. Due to implementation limitations of the framework used, **mapper.tcl** always uses GIF format files.

AI? Request for the named image. Clients may send this if they need an image of the given *name* and *zoom* (as described above for the **AI** command) but no such image is defined yet in the client. This queries the server or connected peers to see if any of them know of the needed image. The client should continue to process tasks without waiting for a response (which is never guaranteed). If another peer knows of the requested image, it should respond with an "**AI**" command. The payload is a JSON object with the following fields:

Name The image name as known to the mapper.

Sizes *(list of objects)*
The list of sizes for which the image data are requested. Each element of the list is an object with the following field:

Zoom *(float)*
The requested magnification factor for the image.

ALLOW The client MAY send this command to the server after logging in. This tells the server that the client supports one or more optional features. The JSON payload has the following field:

Features *(list of strings)*
This is a list of feature names which the client wishes to enable. Every time an **ALLOW** command is given, it resets the entire set of features, rather than adding to them. Currently, only one feature is recognized:

DICE-COLOR-BOXES
The client supports formatting controls in the die-roll title string sent to the server in the **D** command and received from the server via the **ROLL** command. Specifically, the title may consist of multiple sub-titles separated by U+2016 characters (‖). Each of these may also have a suffix consisting of the U+2261 (≡) character followed by a color name or **#***rrggbb* color code, indicating the foreground color for that part of the title. A second such suffix may be added to indicate the background color. If this feature is not enabled, these formatting codes are stripped out by the server before sending die roll results to a client which isn't prepared to interpret them.

AUTH If the server included a challenge string (see the **OK** command below), then it requires a password before the client is allowed to join the server. This is intended to prevent accidental connections by clients to the wrong servers, and to filter out nuisance connections from spammers or other random connections, so it is a fairly simple authentication mechanism using a password shared by all clients in a play group. The payload is a JSON object with the following parameters:

Client This describes the client program. By convention, this value SHOULD include the version number (e.g., "mapper 4.0.1").

Response *(base64-encoded bytes)*
The client's response to the server's challenge.

User If provided, the *user* parameter gives the local username of the player who is running the client. This name will be displayed for purposes such as chat-window conversations and dice rolling. If not provided, "anonymous" will be used, unless the authentication is successful for the GM role, in which case "GM" will be used regardless of any value sent by the client for the *user* field.

Calculation of the *response* field is as follows.

Given:

C is the server's challenge as a binary string of bytes (after decoding from the base-64 string actually sent by the server),

H(x) is the binary SHA-256 hash digest of x,

P is the password needed to connect to the server, and the notation

$x\|y$ means to concatenate strings x and y,

To calculate the *response* string, the client must do the following:

(1) Obtain i by extracting the first two bytes from C as an unsigned, big-endian, 16-bit integer value.

(2) Calculate D=H($C\|P$).

(3) Repeat i times: Calculate D'=H($P\|D$); then let D=D'

The binary value of D is then encoded using base-64 and sent as the *response* value.

Starting with protocol version 332, the server is guaranteed to respond to the **AUTH** command with either a **DENIED** or **GRANTED** message (q.v.). The client MAY wait for this response before proceeding with its other operations, so that it knows for sure how the authentication went.

AV Adjust the map's viewport by making the grid square identified by the **Grid** parameter visible at the top-left of the display. If this field is not present or is empty, then clients MAY fall back to the older behavior of adjusting the view by setting the horizontal scrollbar to *xview* as a fraction of its full distance, where 0.0 is all the way to the left and 1.0 is all the way to the right; the vertical scrollbar is similarly set based on *yview* where 0.0 is all the way to the top and 1.0 is all the way to the bottom. The payload is a JSON object with the following fields:

Grid The name of the grid square which should be in the upper-left of the display. This uses the same naming convention as the grid is labelled on the display (e.g., the far upper-left grid square of the entire map is **A0**).

XView *(float)*
The fraction of full distance the x-axis scrollbar should be moved to.

YView *(float)*
As *xview*, but for the y axis.

CC Clear the chat history. The payload is a JSON object with the following fields:

RequestedBy
The name of the user who initiated this operation, if known. Clients SHOULD NOT set this field; it is set by the server when sending this message to clients.

DoSilently *(bool)*
If true, the client MAY clear its chat history without notifying the user; otherwise it SHOULD indicate the action to the user.

Target *(int)*
If 0 or missing, clear all messages in the history. Otherwise, if positive, clear all messages with *messageID* less than the *target* value. If negative, clear all but the most recent −*target* messages (e.g., a *target* of −50 clears all but the most recent 50 messages).

MessageID *(int)*
This message counts the same as a chat message and is included in the history itself. Thus, the server sets this field to a unique identifier for this message. Clients SHOULD NOT set this field.

CLR Remove object with a given internal ID from the map. The payload is a JSON object with the following field:

ObjID This may be the object ID for the specific object to remove, or "*" to mean all objects should be removed; "E*" to mean all map elements should be removed; "M*" to mean all monster tokens should be removed; "P*" to mean all player tokens should be removed; or it may be a creature name (in the form "[*image-name*=]*name*") which removes a creature token whose display name matches the *objID* value.

CLR@ This command tells the client to "unload" the contents of a map file. In other words, all the elements listed in that file are removed from the map canvas rather than being added to it. The payload is a JSON object with the following fields:

File The pathname or server ID which specifies the map file in question.

IsLocalFile *(bool)*
If true, *file* refers to a local file the client can read directly. Otherwise it is a server ID which can be used to fetch the file from its cache or from the server.

CO Sets the combat mode state in the client. The payload is a JSON object with the following attribute:

Enabled *(bool)*
If true, the client should be in combat (initiative) mode, otherwise it should not.

CONN Update the list of connected peer clients to those described in the JSON payload, which contains the following field. Clients MUST NOT send this command. This supersedes the **CONN**, **CONN:**, and **CONN.** commands from protocol versions prior to 400.

PeerList *(list of objects)*
A list of objects, each describing a single connected client, with the following fields:

Addr The IP address and port the peer connected from.

User The username as provided during authentication.

Client The name of the client program in use.

LastPolo *(float)*
The number of seconds since the server last heard a **POLO** command from the peer.

IsAuthenticated *(bool)*
If true, the client successfully negotiated the server's authentication process.

IsMe *(bool)*
If true, this object describes the client which received this command.

CS Update the client's clock. The JSON payload includes the following fields:

Absolute *(int64)*
The absolute time on the GMA world clock (in tenths of seconds since the epoch).

Relative *(int64)*
The elapsed time on the GMA world clock (in tenths of seconds since the GM set a reference point, e.g., the start of combat).

Running *(bool)*
If true, the local client should continue advancing the clock in real-time locally after updating the time to the new absolute time. Otherwise, cancel real-time updates if they were running, leaving the clock at the given absolute time. The local real-time updating events should only be performed when not in combat mode.

D Roll dice using the server's built-in die rolling facility. The JSON payload contains the following fields:

Recipients *(list of strings)*
The names of the people who should receive the results of the die roll. For global or GM results, this should be **null**.

RequestID
If you put a string value in this field, it will be sent back in any **ROLL** response messages the server sends to you in response to this request. This allows a client to associate responses to the requests that generated them. You may leave this blank or omit it if you don't care to match requests with responses.

ToAll *(bool)*
If true, this is a global die-roll, and the result should be sent to all clients.

ToGM *(bool)*
If true, this die-roll will be sent "blind" to the GM only. Not even the requester will see the result of the roll, only the GM will.

RollSpec
The die-roll specification, like "d20+12" or "6d6 fire". If this field is empty, the previous die-roll from this client is repeated, or "1d20" if there was no previous one.

DD Store Die-Roll Presets. Clients send this command to the server to request it to store a set of die-roll presets for it to retrieve for use in subsequent sessions. The JSON payload is identical to the one sent by the server in the **DD=** command, with the addition of the following optional field:

For The name of the user whose presets are being affected. This is optional; if omitted, the presets of the user making this request are affected. Only the GM may change other users' presets.

The server SHOULD respond by issuing a **DD=** command to all peer clients logged in with the same username as the affected user.

DD+ Add to Die-Roll Presets. Clients send this command to the server to add an additional set of die-roll presets to the collection it was already holding for that user. The JSON payload is identical to the one sent for the **DD** command. The server SHOULD respond by issuing a **DD=** command to all peer clients logged in with the same username as the requesting client.

DD/ Filter die-roll presets. This removes a set of die-roll presets from the server's storage for the user. The server SHOULD respond by issuing a **DD=** command to all peer clients logged in with the same username as the affected user. The JSON payload includes the following field:

Filter The value is a regular expression. The server will delete all die-roll presets whose *name* attribute matches this regular expression.

For The name of the user whose presets are being filtered. This is optional; if omitted, the presets of the user making this request are affected. Only the GM may filter other users' presets.

DD= This command defines the list of die-roll presets the client should know. This replaces any previous set of presets the client was using. Clients MUST NOT send this command. This supersedes the commands **DD=**, **DD:**, and **DD.** in protocol versions prior to 400. The JSON payload includes the following fields:

Presets *(list of objects)*
Each element in this list describes a single preset. It has the following fields:

Name The name by which this preset is identified to the user. This MUST be unique for that user. The client SHOULD sort the preset list by the *name* field before displaying. If a vertical bar ("|") appears in the *name*, all text up to and including the bar SHOULD be suppressed from display.

Description
A text description of the purpose of the preset.

DieRollSpec
The die-roll specification to send to the server when rolling this preset.

DENIED Access to the server is denied. The server will send this to a client; clients MAY NOT send it. The JSON payload sent with this command includes the field:

Reason
Text description of why the access was refused.

DR Retrieve the authenticated user's die roll presets. This SHOULD result in the server sending the **DD=** command back to the requesting client.

DSM Defines (or re-defines) a condition status marker which the client SHOULD use to mark creature tokens when the creature has the associated condition in effect. The JSON payload includes the following fields:

Condition
The name of the condition being described. If there was already a marker defined for that condition, it is replaced with the new definition. If either *shape* or *color* is the empty string or **null**, then the condition is effectively removed from the list of conditions known to the mapper. Creature conditions are in effect if their name appears in the list value for that creature's **STATUSLIST** attribute. The mapper comes with the following conditions pre-defined: **ability drained**, **bleed**, **blinded**, **confused**, **cowering**, **dazed**, **dazzled**, **deafened**, **disabled**, **dying**, **energy drained**, **entangled**, **exhausted**, **fascinated**, **fatigued**, **flat-footed**, **frightened**, **grappled**, **helpless**, **incorporeal**, **invisible**, **nauseated**, **panicked**, **paralyzed**, **petrified**, **pinned**, **poisoned**, **prone**, **shaken**, **sickened**, **stable**, **staggered**, **stunned**, and **unconscious**.

Shape The shape of the marker to be placed if the creature has this condition. The mapper will attempt to arrange multiple markers with the same shape such that they are all visible at the same time. This value may be one of the following:

- **|v** A small downward-pointing triangle against the middle of the left edge of the token. (This is a lower-case "v".)
- **v|** A small downward-pointing triangle against the middle of the right edge of the token. (This is a lower-case "v".)
- **|o** A small circle against the middle of the left edge of the token. (This is a lower-case "o".)
- **o|** A small circle against the middle of the right edge of the token. (This is a lower-case "o".)
- **|<>** A small diamond against the middle of the left edge of the token.
- **<>|** A small diamond against the middle of the right edge of the token.
- **/** A slash (upper right to lower left) through the entire token.
- **** A back-slash (upper left to lower right) through the entire token.
- **//** A double slash (upper right to lower left) through the entire token.
- **** A double back-slash (upper left to lower right) through the entire token.
- **–** A single horizontal line drawn through the center of the entire token.
- **=** A double horizontal line drawn through the center of the entire token.
- **|** A single vertical line drawn through the center of the entire token.

‖ A double vertical line drawn through the center of the entire token.

+ A cross drawn through the entire token.

A hash-mark drawn through the entire token.

V A large downward triangle drawn around the entire token. (This is an upper-case letter "V".)

^ A large upward triangle drawn around the entire token.

<> A large diamond drawn around the entire token.

O A large circle drawn around the entire token. (This is an upper-case letter "O".)

Color The color to draw the marker in any of the forms documented above, or the special value "*", which means to draw the marker in the same color as the creature's threatened area.

If *color* begins with "**−−**" (e.g., "−−red"), then the marker is drawn with dashed lines instead of solid ones. If it begins with "**..**" (e.g., "..blue"), then the effect is the same, but the dashes are shorter.

Description
A description of the effects on the character of having that condition applied. This is intended to be shown to players (for example, if they hover their mouse over an affected creature token).

ECHO Ask that the server send this message back to you. This may be used for synchronization since it indicates to the client when the server got to this request amongst the others it received. The following fields may be populated:

s An arbitrary string value.

i *(int)* An arbitrary integer value.

b *(bool)* An arbitrary boolean value.

o *(object)* An arbitrary set of key/value pairs, with each value having any type.

GRANTED Access to the server is granted. The server will send this to a client; no client should send it. The JSON payload contains the field:

User The user name for this client. This is the name provided by the client, "anonymous" if the client didn't give one, or "GM" if the client authenticated as the GM.

I Update the time clock for initiative-based actions. If the mapper is in combat mode, the clock display is updated accordingly. The JSON object payload includes the following fields:

ActorID
The object identifier of the creature whose turn it is. This may be the unique object ID code, the creature name as documented in the **AC** command, the special string "*Monsters*" which indicates that all creatures with monster-type tokens have initiative, or may be of the form "**/***regex*", which matches all creatures whose names match the regular expression *regex*.

Hours *(int)*
The number of hours elapsed since the start of combat.

Minutes *(int)*
The number of minutes elapsed since the start of this hour of combat.

Seconds *(int)*
The number of seconds elapsed since the start of this minute of combat.

Rounds *(int)*
The number of rounds elapsed since the start of combat.

Count *(int)*
The number of initiative slots elapsed since the start of the round.

IL Update the initiative list. The JSON payload includes the following field:

InitiativeList *(list of objects)*
The current initiative list is given as a list of initiative slots, each of which is an object with the following fields:

Slot *(int)*
The slot number. As currently implemented this is a number in the range [0,59] which gives the "count" (1/10th second) in the round where this creature may act.

CurrentHP *(int)*
The creature's current hit point total.

Name The creature's name as displayed on the map.

IsHolding *(bool)*
If true, the creature is holding their action.

HasReadiedAction *(bool)*
If true, the creature is holding a readied action.

IsFlatFooted *(bool)*
If true, the creature is flat-footed.

L Loads map elements into the map client, either replacing or adding to the current contents of the map. Supersedes the function of commands **L**, **M**, **M?**, and **M@** in protocol versions prior to 400. The payload is a JSON object with the following fields:

File The local pathname or server ID identifying the map file to be loaded.

IsLocalFile *(bool)*
If true, *file* refers to a local filename; otherwise it is a server ID.

CacheOnly *(bool)*
If true, the server is only advising the client that it would be good to have a cached copy of *file* on hand for later. The client MUST NOT actually load the file's contents to the displayed map at this time.

Merge *(bool)*
If true, the map elements in *file* are merged with the map's current contents. Otherwise the map's current elements are replaced by the new ones in *file*.

LS-*type* This set of commands load a single map element into the map. As opposed to the **L** command which directs the client to read a file to get one or more objects, this just sends an object directly to it. This may be used, for example, when one client draws an element interactively and wants the other clients to display it as well.

These supersede the function of the commands **LS**, **LS:**, and **LS.** in protocol versions prior to 400.

All of the following **LS-***type* commands include as many of the following parameters as are applicable to them, in addition to type-specific parameters:

ID The unique object identifier. This is a string containing upper- or lower-case letters, digits, underscores and octothorpes. By convention, we create these as hexadecimal UUID values, but they may be any arbitrary string, including human-readable IDs such as "PC1", etc.

X *(float)*
The *x* coordinate of the "reference point" of the element, in standard map pixel units.

Y *(float)*
The *y* coordinate of the "reference point" of the element, in standard map pixel units.

Points *(list of coordinate objects)*
If an object needs more than a single coordinate pair to specify their location, (e.g., the diagonally opposite corner of a rectangle) the subsequent points are listed in this parameter. Each element in the list is an object with the fields:

X *(float)*
The *x* coordinate in standard map pixel units.

Y *(float)*
The *y* coordinate in standard map pixel units.

Z *(int)* The *z* "coordinate" of the element is its vertical stacking order on the displayed map canvas. Higher numbers are drawn after lower numbers. If two objects have the same *z* value and physically overlap, the result is not defined.

Line The color used to draw the shape's outline, as a standard color name or RGB string such as "#336699".

Fill As with the *line* field, specifies a color to fill the interior of the element. If omitted or empty, the object will not be filled. Note that line objects are *filled* with the *fill* color. They don't have an outline so don't use the *line* value.

Width *(int)*
The width in pixel units of the element's outline.

Layer The map layer this element belongs to. Currently not implemented.

Level The dungeon level where this element appears. Currently not implemented.

Group The object group to which this element belongs. Currently not implemented.

Dash *(int)*
The outline of the element is to be drawn with the specified dash pattern:

0 Solid (the default)

1 Long dashes

2 Medium dashes

3 Short dashes

4 Long-Short pattern

5 Long-Long-Short pattern

Hidden *(bool)*
If true, this element MUST NOT be displayed on-screen.

Locked *(bool)*
If true, this element MUST NOT be edited further by clients.

Each of the following commands which begin with **LS-** defines a different kind of map element.

LS-ARC
Draws an arc on the canvas. The arc is drawn around the circumference of the ellipse inscribed in the rectangle defined by the reference point and the first point in the *points* attribute (as opposite corners of a rectangle). The payload is a JSON object with these parameters in addition to those common to all map elements:

ArcMode *(int)*
Specifies how to draw the arc on-screen.

0 Pie slice (default); connects the arc endpoints to the center of the circle.

1 Arc; does not connect the endpoints to anything else.

2 Chord; connects the endpoints to each other via a straight line segment.

Start *(float)*
The number of degrees around the circle to begin drawing the arc.

Extent *(float)*
The number of degrees around the circle to end drawing the arc.

LS-CIRC
Draws an ellipse on the map canvas. The payload is a JSON object as described for all map elements above. The ellipse is defined as described for the **LS-ARC** command, but the entire circumference is drawn.

LS-LINE
Draws a straight line segment from the reference point to the each point in the *points* attribute (if more than one point is in *points* the result will be multiple connected line segments). The JSON payload has the following attribute in addition to the common ones described above:

Arrow *(int)*
The style of arrows to draw on the ends of the line:

0 No arrows

1 Arrow on the first point (the reference)

2 Arrow on the last point

3 Arrows on both first and last points

LS-POLY
Draws a polygon on the map canvas. The vertices of the polygon start at the reference point and continue through every point in the *points* attribute, and finally connecting back to the reference point again. The JSON payload contains the following fields in addition to the common ones described above:

Spline *(int)*
The factor to use when smoothing the sides of the polygon between its points. 0 means not to smooth at all, resulting in a shape with straight edges between the vertices. Otherwise, larger values provide more smoothing.

Join *(int)*
The join style for the edges of the polygon:

0 Beveled corners

1 Mitered corners

2 Rounded corners

LS-RECT
Draws a rectangle defined by the reference point and the first point in the *points* attribute (as opposite corners of the rectangle).

LS-SAOE
Draws the zone of a spell's area of effect on the canvas. The JSON payload contains the following field in addition to those described above:

AoEShape *(int)*
The shape of the area of effect:

0 Cone: a 90° pieslice defined as described for a pieslice arc element.

1 Radius: an ellipse defined as described for a circle element.

2 Ray: a rectangle defined as described for rectangle elements.

LS-TEXT
Places some text on the canvas. The JSON payload includes the following fields in addition to those common to all elements:

Text The text to be displayed.

Font *(object)*
The typeface to use for the text. This is an object with the following fields:

Family The name of the font family as recognized by Tk.

Size *(float)*
The font size as recognized by Tk.

Weight *(int)*
The type wight to use. This may be 0 for normal or 1 for boldface.

Slant *(int)*
The type slant to use. This may be 0 for Roman (normal) or 1 for Italic (slanted).

Anchor *(int)*
Where the reference point is considered to be relative to the text:

0 Center

1 North

2 South

3 East

4 West

5 Northeast

6 Northwest

7 Southwest

8 Southeast

LS-TILE
Draws an image tile on the canvas. The JSON payload has the following fields in addition to those common to all map elements:

Image The image name as known to the mapper.

BBHeight *(float)*
The bounding-box height for the region bordering the image.

BBWidth *(float)*
The bounding-box width for the region bordering the image.

MARCO

Status check. If received, reply with a **POLO** command. Clients MUST NOT send this command.

N.B. It is important that your client not ignore these occasional ping messages. For example, if your client is too slow receiving messages such that the server needs to expend extra work to queue them up for you, it will be willing to do so if you have been at least responding to **MARCO** messages. If you haven't, the server will suspect your client has locked up or is not going to be able to catch up with the data being sent to it, and may decide to terminate the client's connection.

MARK

Make a brief animated marker at the specified coordinates to draw attention to that space. The JSON payload contains the following fields:

X *(float)*
The *x* coordinate of the marker.

Y *(float)*
The *y* coordinate of the marker.

OA Updates attributes of a specified map object. Any attributes not listed in the command MUST remain as-is. The JSON payload includes the following fields:

ObjID The unique object identifier for the object to be modified. Alternatively, it may be in the form @*name* where *name* is the creature's name in any of the forms allowed for the **AC** command.

NewAttrs *(object)*
The set of new attributes and their values, expressed as an object with field names matching the object attributes to be changed and their associated values being the new values for those attributes.

Example: **OA {"ObjID":"a984a3","NewAttrs":{"X":12,"Y":44.5}}**

Note that there was an implicit assumption in the past that the **NAME** attribute of a creature would not change and it could be used as an immutable identifier within a map client for a creature token. However, this is not the case and in fact GMA now includes explicit features to change this very attribute. Clients must be prepared to deal with the consequences of a change to any attribute, including **NAME**. The **mapper** client itself does this correctly starting with version 3.39.

OA+ Adds one or more values to a single attribute of an object known to the mapper. The payload is a JSON object with the following fields:

ObjID The unique object identifier, specified as for the **OA** command.

AttrName
The name of the attribute to be modified. This MUST be an attribute whose value is a list of strings. (E.g., the **STATUSLIST** attribute of creature objects.)

Values *(list of strings)*
A list of string values to be added to those already in the named attribute.

OA− Removes one or more values from a single attribute of an object known to the mapper. The payload is a JSON object with the following fields:

ObjID The unique object identifier, specified as for the **OA** command.

AttrName
The name of the attribute to be modified. This MUST be an attribute whose value is a list of strings. (E.g., the **STATUSLIST** attribute of creature objects.)

Values *(list of strings)*
A list of string values to be removed from those already in the named attribute. It is not an error if the value wasn't there to begin with.

OK The server sends this to the client when it is ready for the client to start sending commands to it. The accompanying JSON payload includes the following fields:

Protocol *(int)*
The protocol version used by the server. Map clients which do not support that protocol SHOULD warn their users and SHOULD disconnect since they can't guarantee they can actually communicate with the server.

Challenge *(base64-encoded bytes)*
If present, the *challenge* value is a base-64-encoded authentication challenge. A client must successfully respond with a valid **AUTH** command before any of its commands to the server will be accepted. The server will also refuse to send the client any map updates until it has successfully authenticated.

PRIV Notice from the server that a command sent by the client is denied due to insufficient privilege. Clients MUST NOT send this command. The JSON payload includes the following field:

Command
The command the client was attempting.

Reason
The reason that command was denied.

POLO No-op. Clients MUST ignore if received. Clients SHOULD send this in response to a **MARCO** command from the server.

PROGRESS
The server sends this to the client to indicate progress of a long-running operation such as downloading files. (Formerly this was sent as a comment prior to protocol version 400.) The payload is a JSON object with the following fields:

OperationID
A unique identifier for the operation we're reporting progress for.

Title The description of the operation in progress. This is suitable for display to the user.

Value *(int)*
The current value of the progress meter. Units are arbitrary.

MaxValue *(int)*
The maximum value expected for the progress meter when it is finished. If this is 0, we don't yet know what the maximum will be, which SHOULD cause the client to use a progress meter style that indicates that it is not possible to estimate remaining time to completion. Also note that this value MAY change over the course of the progress report if the server becomes aware that it has a better maximum value available to it at that point.

IsDone *(bool)*
A boolean indication that the operation is completed. The client SHOULD remove the progress meter from display when the operation is completed.

Clients SHOULD NOT send these; they are for the server to notify the client about progress. If a client does send progress update messages to other peers, it should be clear that what is being tracked is a process that the other clients are interested in.

PS Place a creature token on the map. This may be used to define a new creature object if no object with the given *ID* already exists, or replaces an existing token with the (possibly different) parameters given. The JSON payload includes the following fields:

ID The internal ID by which this creature is to be known. This must be unique. The client which creates the character token locally should create a unique ID, which is then broadcast via this command to the other clients, which use the same ID.

Name The name as displayed on the map. Must be unique among all creatures currently displayed.

Health *(object)*
If not **null**, this gives the current health status of the creature. It is an object with the following fields:

MaxHP *(int)*
The creature's maximum number of hit points.

LethalDamage *(int)*
The amount of lethal damage sustained.

NonLethalDamage *(int)*
The amount of non-lethal damage sustained.

Con *(int)*
The grace amount of hit points the creature may sustain over their maximum before they are considered dead.

IsFlatFooted *(bool)*
If true, the creature is flat-footed.

IsStable *(bool)*
If true, the creature has been stabilized to prevent death while critically wounded. Creatures which are in non-critical states of health don't have this attribute set even though technically (in a sense) they are "stable".

Condition
A custom condition status to display on the map, if you wish to override the map's calculation.

HPBlur *(int)*
If 0, the creature's health is displayed accurately. Otherwise, this gives the percentage by which to "blur" the hit points as seen by the players. For example, if *HPBlur* is 10, then hit points are displayed only in 10% increments.

Gx *(float)*
The grid *x* coordinate for the creature's reference point. Note that this is in grids, not pixel units.

Gy *(float)*
The grid *y* coordinate for the creature's reference point. Note that this is in grids, not pixel units.

Skin *(int)*
If 0, show the default appearance on the creature's token. Otherwise, show one of the alternative images defined for that creature.

SkinSize *(list of strings)*
: This gives a list of values to use for the *size* attribute corresponding to the *skin* number. For example, if there are 3 skins defined for a shape-changing creature, and the first two are medium-size but the third is large, then *skinsize* would have the value **["M","M","L"]**.

Elev *(int)*
: The creature's elevation in feet relative to the "floor" level.

Color
: The color code used to draw the creature's threat zone.

Note
: A note to attach to the creature token to indicate special conditions or status.

Size
: The tactical size category of the creature.

Area
: The tactical size category for the creature's threat zone.

StatusList *(list of strings)*
: A list of condition codes which apply to the character. These are defined by the **DSM** command (q.v.).

AoE *(object)*
: If not **null**, a spell area of effect should be drawn around the creature. The value is an object with these fields:

  Radius *(float)*
  : The distance in standard map pixel units away from the creature's center to the perimeter of the area of effect.

  Color
  : The color with which to draw the spell area.

MoveMode *(int)*
: The mode of locomotion currently employed by the creature:

  0 Land (walking)

  1 Burrowing

  2 Climbing

  3 Flying

  4 Swimming

Reach *(int)*
: Indicates if the creature is wielding a reach weapon and thus has an expanded threat zone. If 0, the threat zone is normal for the creature. If 1, it cannot attack adjacent foes but has a wider threat space. If 2, it has both threat zones active at the same time.

Killed *(bool)*
: If true, the creature is dead.

Dim *(bool)*
: If true, the creature does not have initiative now, and their token should be de-emphasized compared to the one with initiative.

CreatureType *(int)*
: The specific type of creature. This may be 0 if the type is unknown, 1 for monsters, or 2 for players. Clients MUST NOT set this field to a value other than 1 or 2.

READY
: The server sends this to the client to indicate that all preliminary data has been sent, authentication (if applicable) has been successful, and the client may proceed to

carry out normal operations now.

ROLL Reports the results of a die roll initiated by the **D** command. Clients MUST NOT send this command. The JSON payload includes the following fields:

Sender The name of the user who rolled the dice.

Recipients *(list of strings)*
The names of the people the message was explicitly addressed to. For global messages, this should be **null**.

MessageID *(int)*
The unique number assigned by the server for this chat message. (Die roll results are essentially a kind of chat message.)

MoreResults *(bool)*
If true, there will be more **ROLL** messages yet to come which are part of the result set for the same die roll request.

RequestID
The **RequestID** string passed by the client when requesting this die roll, or the empty string if they didn't.

ToAll *(bool)*
If true, this is a global message sent to all clients. The contents of the *recipients* field are ignored; it should be omitted or set to **null**.

ToGM *(bool)*
If true, this message will be sent only to the GM. The contents of the *recipients* field are ignored; it should be omitted or set to **null**.

Title The title describing the purpose of the die-roll as set by the user, if any. The title string may include special formatting codes which will be used by clients supporting the **DICE-COLOR-BOXES** optional feature (see the **ALLOW** command for details). Clients which did not request this feature will not see the special formatting codes in the title string.

Result *(object)*
The result of the die roll. This is an object with the following fields:

Result *(int)*
The final numerical result of the die-roll.

InvalidRequest *(bool)*
If true, the die roll never happened because the request could not be understood. In this case, the details about the request and reason for failure are in the **Details** array. The **Result** integer field should be disregarded since there was no result generated.

ResultSuppressed *(bool)*
If true, this result message describes a die roll request without disclosing the results of the roll. This is used, for example, when sending a die roll for the GM to see privately (not even showing the requester the result). In this case, the value of the **Result** field should be ignored.

Details *(list of objects)*
The details behind how that result was generated are given as a list of objects with the following fields:

Type A text label describing what the value means in the context of the die-roll result.

Value The value for this part of the result (as a string).

SYNC This command MUST be ignored if received by a client. A client sends this to the server to request a replay of map commands which would be necessary to catch it up to the state the other maps are currently in. A server may be configured to perform a sync operation upon client connection (after, if required, authentication) without the client explicitly sending a **SYNC** command to it.

SYNC-CHAT

This command MUST be ignored if received by a client. Supersedes the **SYNC** command with **CHAT** argument in protocol versions prior to 400. A client sends this to the server to request a replay of chat messages (including die-roll result notices) according to the JSON payload, which may include this field:

Target *(int)*

If missing or 0, all chat messages in the server's history are sent to the client. Otherwise, if *target* is positive, all messages with *messageID*s greater than *target*. If *target* is negative, its absolute value gives the number of recent messages to send to the client (e.g., if *target* is −50, then the most recent 50 messages are re-sent to the client).

TB Controls the display of the toolbar in the mapper client. The JSON payload is an object with the following field:

Enabled *(bool)*

If true, the client SHOULD display its toolbar. If false, the client SHOULD NOT display it. The client may override this request based on the user's preferences.

TO Send chat message to a set of recipients. The payload is a JSON object with the following fields:

Sender The name of the user sending the message. Clients SHOULD NOT set this field; it is set by the server when sending these messages to peers.

Recipients *(list of strings)*

The names of the people the message was explicitly addressed to. For global messages, this should be **null**.

MessageID *(int)*

The unique number assigned by the server for this chat message. Clients SHOULD NOT set this field.

ToAll *(bool)*

If true, this is a global message sent to all clients. The contents of the *recipients* field are ignored; it should be omitted or set to **null**.

ToGM *(bool)*

If true, this message will be sent only to the GM. The contents of the *recipients* field are ignored; it should be omitted or set to **null**.

Text The text of the message to be sent.

/CONN

For diagnostic purposes, this command causes the server to send a summary of the connections it is currently serving. These are sent back to the client via the **CONN** command documented above.

UPDATES

This provides the latest available versions of various programs. A client MAY alert the user or initiate an automatic upgrade if it recognizes that it or one of its dependencies is out of date. Clients MUST NOT send this command. (Formerly this was sent as a comment prior to protocol version 400.) The payload contains the

following field:

Packages *(list of objects)*
Each element in this list is a JSON object with the following fields:

Name The name of the software package, such as “mapper”, “go-gma”, or “core”.

Instances *(list of objects)*
A list of all available instances of the named package. Each is a JSON object with the following fields:

OS The name of the operating system platform for which this instance of the package is available (e.g., “freebsd”, “linux”, “darwin”, “windows”, etc.). If this is empty, then this instance is platform-independent.

Arch The name of the hardware architecture for which this instance of the package is compiled (e.g., “amd64”, etc.) If this is empty, then this instance is architecture-independent.

Version
The semantic version number that users should be using. This is set by the GM (or designated server administrator—henceforth we will simply use the term “GM” for this role) for their game. It may not necessarily be the latest version available generally.

Token The download token used for obtaining a copy of the package. This is entirely under the GM’s control and refers to a file the GM placed on their game server for their players to access. The *token* value is the non-static part of the download URL from their game server.

WORLD
This command provides various campaign-world information useful for the client to know. Clients MUST NOT send this command. The payload includes the following parameters:

Calendar
The name of the calendar system in play, as known to the GMA game clock. (Formerly this was sent as a comment in the form **// CALENDAR // *system*** prior to protocol version 400.)

Declares which calendaring system the various dates and times are based upon.

Example: **WORLD {"Calendar":"golarion"}**

WEIRD SIZES

While the mapper implements the standard d20/Pathfinder creature size categories, including tall (uppercase) and wide (lowercase) variants, sometimes there are special cases which fall outside that list. The following special codes are also usable:

M20/m20 Medium creature (5-foot space, 5-foot threat zone) with a 20-foot reach zone.

L0/l0 10-foot swarm (10-foot space, no threat or reach zone).

INSTALLATION

To run the mapper, you’ll need an up-to-date Tcl/Tk interpreter (8.6 or later), and the following packages in your Tcl runtime library:

- uuid
- base64
- struct::queue
- tklib

Additionally you will need a copy of **curl**(1) installed on your system.

If you will be uploading content to the web server (and are authorized to do so), you will also need to have **ssh**(1) and **scp**(1) on your system.

You will need to ensure that the paths to these commands, server name(s), data paths, etc, are configured correctly for your needs as well.

SEE ALSO

curl(1), **scp**(1), **ssh**(1), **dice**(3), **dice**(5), **mapper**(5), **gma**(6).

The wiki page **https://www.madscience.zone/gamers/wiki/GmaSoftwareDevelopment/Protocols/MapperProtocol** contains more detail on the map protocol, but this manpage is the definitive reference to the protocol.

AUTHORS

Steve Willoughby / steve@madscience.zone; John Mechalas (elevation and movement modes).

HISTORY

This document describes version 4.x of **mapper**. This introduces a breaking change from versions 3.x, mostly in terms of the communications protocol used between the mapper server and clients and the way map data are represented internally and in the disk files used by the mapper.

A version of **mapper** was also in version 3 of GMA, but was different in operation.

As of version 3.25, the operation of the "Push to other clients" button was changed so that it only works in store-and-forward mode (and is thus reserved essentially for privileged users only). This was done because the old function of that button is no longer needed and tended to cause more trouble than it was worth anyway.

Also changed in 3.25 is the function of the "Sync" button. It used to simply attempt to reconnect a client to the server (which should automatically happen anyway). Now, since the server tracks game state, simply exiting and restarting the map client accomplishes the same effect (possibly better). Now this button requests a "sync" operation with the server.

This document also describes map protocol version 402, which improved the **ROLL** server message by adding support for blind rolls to the GM.

Protocol version 401 introduced the **Grid** parameter to the **AV** command.

Protocol version 400 changed the server communication protocol to use JSON instead of TCL lists. It also introduced the **PROGRESS**, **PROTOCOL**, **READY**, **UPDATES**, and **WORLD** commands.

As of the beta-3 release, the **RequestID** and **MoreResults** fields were added to die roll requests and results messages.

Version 332 added the **GRANTED** server response, guaranteeing that **AUTH** will be followed by a response in any case.

Version 331 changed the way persistent chat messages are managed, altering the **ROLL**, **TO**, **SYNC**, and **CC** commands.

Version 329 added the **CC** command and the extended form of **SYNC**.

Previously, version 328 introduced the **ACCEPT** command.

Version 327 added the **DENIED** and **PRIV** response commands.

Version 326 added support for chat channels and die rolling, and changed the response from the **/CONN** command to be a structured sequence of data records rather than sending the reply in comments.

Protocol 325 added connection debugging support by changing the **AUTH** command and adding **/CONN**.

The version 324 protocol added the **DSM**, **OA+**, and **OA−** protocol commands. This version of the protocol also assumes mapper file format 15 is being used.

Previously, protocol version 323 added support for server-side persistent state and the introduction of the **SYNC** protocol command.

Protocol version 322 added comments to version 321.

Protocol version 321 added authentication support to the capabilities of protocol 320.

Map protocol version 320 has the same commands as 319, but the **HEALTH** attribute of creatures changed from map version 12 to 13, so we are advancing the protocol version at the same time since maps released for protocol 319 were expecting the older map format.

Map protocol version 319 differs from version 318 (the first explicitly numbered version) by the addition of the **CLR@**, **M?**, and **M@** commands.

COMPATIBILITY

This program requires a reasonably modern version of Tcl/Tk, tcllib and tklib to function properly. We strongly recommend running it with the latest versions of all of those.

It is known to run on the macOS Mojave platform (tested on 10.14.6), macOS Catalina (tested on 10.15.3, but note that Apple's support for python3, tcl, and tk is such that you may want to install your own versions of those tools); and should run fine on any modern *NIX-like platform (tested on FreeBSD 12.0, Ubuntu 18.04 LTS, and Ubuntu 16.04 LTS).

It was also tested (briefly) on Windows 10. Specific notes about using it on that platform are in the following section.

WINDOWS USAGE

To install on Windows, you will need to install the following products in addition to GMA:

- A Tcl/Tk interpreter. We tested with Active Tcl 8.6 from *activestate.com*.
- The Curl utility. We tested with Curl 7.67.0 from *curl.haxx.se*.
- The SSH shell program, including the SCP command for copying files (if you want to use the store and forward mode to send data to the server; note that this requires authorization to perform that action by the server administrator and that you have configured the SSH/SCP program(s) with the valid credential certificates). We tested with PuTTY 0.73 from *www.chiark.greenend.org.uk*.

By default, the **mapper** program will try to load its configuration file from *homedir***\.gma\mapper\mapper.conf** if that file exists, where *homedir* is the user's home directory. This allows the mapper to be launched by double-clicking its file icon without needing to provide command-line options.

We recommend that you set up this configuration file before starting, since we don't currently have a fancy Windows installer wizard thingy to configure it for you.

In this file, you will need at a minimum to tell the mapper where to find the **curl** program and the network locations in use for your game. For example:

curl-path=C:\curl-7.67.0-win64-mingw\bin\curl.exe
curl-url-base=https://www.example.org/game/map
host=www.example.org
port=2323
proxy-url=http://proxy.example.org:1080

If your game does not use a network server at all, it's possible you won't need a configuration file.

Handling of standard output and standard error devices for printing messages is not as well-supported in Windows as in Unix-like operating systems, so the mapper will attempt to place output in files called **mapper.***pid***.stdout** and **mapper.***pid***.stderr** in the *homedir***\.gma\mapper\logs** directory. If the mapper just silently fails to start, look in those to see if there were fatal error messages recorded there.

The mapper does not currently support being the forwarder in a store and forward configuration for Windows clients.

FILES

~/.gma/mapper/mapper.conf
Default configuration file read if no explicit **−C** or **−−config** option is given.

~/.gma/mapper/style.conf
Default custom style configuration file read if no explicit **−s** or **−−style** option is given.

~/.gma/mapper/debug.log
Location where debugging messages are written in addition to being displayed in the debugging window.

~/.gma/mapper/logs
Runtime logfiles are stored here for each execution of the mapper.

~/.gma/mapper/cache
Cached copies of images and other content are stored here to improve speed of the mapper.

BUGS

There are numerous hacks in the program which really should not be there. In fact, at this point the thing just needs to be rewritten using the newer GMA code base.

Calculation of threatened spaces needs to take elevation into account. (Although the mapper now includes a 3D-aware distance calculation when requested.)

The **−h** option really should have been for **−−help** to conform to usual command-line conventions, and the **−−host** option should instead have been **−H**. This may change in the future.

In previous versions, **−−keep−tools** (**−k**) was called **−−master** (**−m**), but this never really made sense, as it didn't really mean the mapper was in any sort of controlling or leadership role. It only meant it would refuse to turn off its own toolbar if asked to do so. The new name is more descriptive of the actual function.

COPYRIGHT