

GMA Game Master's Guide

6.36.1

www.MadScience.Zone/gma

The information in this document, and the software it describes, are (as of this writing) unreleased works created as an educational exercise and as a matter of personal interest for recreational purposes, shared privately with a few friends.

It is not to be considered an industrial-grade product. It is assumed that the user has the necessary understanding and skill to use it appropriately. The author makes NO representation as to suitability or fitness for any purpose whatsoever, and disclaims any and all liability or warranty to the full extent permitted by applicable law. It is explicitly not designed for use where the safety of persons, animals, property, or anything of real value depends on the correct operation of the software.

For GMA Core software version 6.36.1,
GMA Mapper software version 4.33.1,
and Go-GMA software version 5.29.0.

Copyright © 2022–2025 (some portions 1992–2022) by Steven L. Willoughby (aka MadScience-Zone, previously aka Software Alchemy), Aloha, Oregon, USA. All Rights Reserved. This document is released under the terms and conditions of the Creative Commons "Attribution-NoDerivs 3.0 Unported" license. In summary, you are free to use, reproduce, and redistribute this document provided you give full attribution to its author and do not alter it or create derivative works from it. See http://creativecommons.org/licenses/bynd/3.0/ for the full set of licensing terms.

Pathfinder® is a trademark of Paizo Inc.

Dungeons & Dragons®, D&D®, AD&D®, and Dungeon Master® are registered trademarks of Wizards of the Coast, Inc., a subsidiary of Hasbro, Inc.

Blades in the Dark™ is a trademark of One Seven Design.

Hero Lab® is a registered trademark of LWD Technology, Inc.

Windows® is a registered trademark of Microsoft, Inc.

macOS® is a registered trademark of Apple, Inc.

FreeBSD® is a registered trademark of The FreeBSD Foundation.

PostScript® is a registered trademark of Adobe Systems incorporated in the United States and/or other countries.

ImageMagick® is a registered trademark of ImageMagick Studio LLC.

Unix® is a registered trademark of The Open Group.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Other product names mentioned may be trademarks or registered trademarks of their respective owners; use of such names without mention of trademark status should not be construed as a challenge to such status.



Contents

Contents iii

List of Figures x

List of Tables xii

Preface xiii

1 Introduction 1
- Timekeeping . . . 2
- Maps . . . 3
- Creatures . . . 3
- Game System Support . . . 4
- API . . . 4
- How to Read This Document . . . 4

2 Installation and Setup 7
- Platform . . . 7
- Pre-release Implications . . . 8
- Prerequisites . . . 8
- Installing GMA . . . 10
- Installing GMA Mapper . . . 11
- Installing Go-GMA . . . 11
- Environment Variables . . . 12

3 Unboxing: Diving Right In 13
- Configuring GMA Preferences . . . 14
- Scheduled Events (Timers) . . . 23
- Counters . . . 28
- Drains . . . 29
- Spells . . . 31
- Running Combat . . . 32
- Health Trackers . . . 36
- Managing Initiative and Events in Combat . . . 38
- Ending Combat . . . 44

Condition Timers 44
Endgame 46
Calendar Type 46

4 Introducing the Mapper **49**
Creating Basic Maps 50
A Simple Dungeon Map 62
Using Maps During Combat 62
Spell Areas 65
Measuring Distances 66
Mapping Piecemeal 66
Offline Manipulation of Map Files 70

5 Leveling Up: Adding Graphics to Your Maps **73**
How To Use Creature Tokens 73
How To Use Map Tiles 82
Full Dungeon Map Images 83
Using Map Images From High-Resolution and/or PDF Files . . 94

6 Character Record Sheets **97**
Editing a Character With Charsheet 98
Editing The `.gcr` Data File Directly 99
Printing a Paper Character Sheet 115
Party Skill Summaries 116

7 Roll Initiative: Preparing Encounters in Advance **123**
Basic Encounters and the `.gmo` File 123
Printing and Using the Encounter Sheet 125
Individuals and Groups 131
Random Name Generation 134
Generic Creature Names 138
How Do Names Really Work? 141
Duplicate Nametags 142
A Random Comment About Randomness 142
Hit Points 142
Advanced Customization 144
Attack and Damage Roll Syntax 146
Polymorphing and Shapechanging 147
Custom Sizes 149
Initial Monster Conditions 149
Permanent Invisibility 150
Basing One Monster On Another 150
Automating Combat Setup with the GM Console 151
Ad-Hoc and Staggered Monster Placement 154
Entering and Exiting Combat 155
Extra Non-Combatant Creatures 157

Controlling Printed Output 158

8 Tracking Progress **159**
Failure is an Option 162
Consecutive Successes 167
Pools of Resources 167
Progress Clocks for Monsters 167

9 Bonus Feat: Connecting GMA and the Mapper **171**
Go-GMA 171
Server Basics 172
Setting up a Simple Server 172
Setting up Authentication 174
Really Automating Combat with the GM Console 175
Setting up the Server for Multi-User Access 180
Using the Mapper Console Client 183
Reconnecting to the Server 183
Tracking Hidden Creatures 185
Shifting Loyalties (or at least colors) 185

10 Diplomacy Check: Chat Channels and Die Rolling **187**
Chat Feature in the Mapper Clients 187
Rolling Dice 192
Fancy Things You Can Do With Each Die Roll or Constants . . 193
Global Modifiers 193
Percentile Checks 194
Permutations 195
Blind Rolls 195
Re-Rolling Recent Rolls 196
Pre-set Rolls 196
Die-Roll Modifiers 199
System-Wide Presets and Modifiers 201
Random Lookup Tables 201
Colored Die-Roll Titles 203
Delegating Die Rolls to Other Players 205
Web-Based Die Roller 206
Automatic Rolls for Monsters 207

11 Epic Level: Full Multi-User Server Environment **209**
Setting Up the Web Server 210
Document Directory Layout 211
GM Access (Pushing Content to the Server) 211
Example Play Using the Mapper in Store-and-Forward Mode . 215
Client-Side Caching 217
Automatic Upgrades 217
Server-Side Client Configuration 221

A Command-Line Utility Reference **223**
gma-lockfile(1) 225

B API Function Reference **227**
gma-abilityscore(3) 229
gma-alignment(3) 231
gma-all_culture_names(3) 233
gma-armour(3) 234
gma-authenticator(3) 238
gma-centimeters_to_feet(3) 241
gma-class(3) 242
gma-culture_class(3) 245
gma-default_core_db_path(3) 245
gma-default_preferences_path(3) 245
gma-dice(3) 246
gma-dicepresetcollection(3) 267
gma-dieroller(3) 267
gma-eventcollection(3) 267
gma-feet_to_centimeters(3) 267
gma-from_roman(3) 267
gma-grams_to_pounds(3) 267
gma-hexdump(3) 268
gma-htmlformatter(3) 270
gma-inches_to_centimeters(3) 270
gma-init_application(3) 270
gma-load_map_file(3) 270
gma-load_map_meta_data(3) 270
gma-lockfile(3) 271
gma-map_label_to_xy(3) 273
gma-mapcharacter(3) 273
gma-mapcondition(3) 273
gma-mapevent(3) 273
gma-mapioclientconnection(3) 273
gma-mapiodispatcher(3) 273
gma-mapobject(3) 274
gma-mapper(3) 281
gma-mapservice(3) 284
gma-markupline(3) 297
gma-markuptext(3) 298
gma-markuptextformatter(3) 305
gma-metricize_length(3) 305
gma-metricize_weight(3) 305
gma-name_generator_factory(3) 305
gma-nullformatter(3) 305
gma-ounces_to_grams(3) 305
gma-pfnamegenerator(3) 306

gma-pounds_to_grams(3) 309
gma-preferences(3) 310
gma-ps_str(3) 317
gma-ps_str_raw(3) 317
gma-psformformatter(3) 317
gma-read_map_file(3) 317
gma-read_map_meta_data(3) 317
gma-romannumerals(3) 318
gma-save_map_file(3) 319
gma-simpleconfigfile(3) 320
gma-socketinterface(3) 322
gma-strip_ceature_base_name(3) 341
gma-tcllist(3) 342
gma-to_roman(3) 345
gma-ustar(3) 346
gma-versionutils(3) 350
gma-weightsandmeasures(3) 351
gma-write_map_file(3) 353
gma-xy_to_map_label(3) 353
gmacolors(3) 354
gmafile(3) 355
gmaproto(3) 362
gmautil(3) 371

C File Format Reference **375**
gma-character-record(5) 376
gma-dice(5) 423
gma-mapper(5) 426
gma-monster-sheet(5) 433
gma-style.conf(5) 438
gma.conf(5) 443

D Game Program Reference **449**
gma-crxp(6) 450
gma-forumpost(6) 451
gma-initiative-clock(6) 456
gma-latex-macros(7) 458
gma-loot(6) 459
gma-map-console(6) 463
gma-map-lock(6) 465
gma-map-reduce(6) 466
gma-map-translate(6) 468
gma-mapper(6) 470
gma-mapservice(6) 494
gma-markup(6) 498
gma-mkmap(6) 500

gma-modcleanup(6) . . . 504
gma-namegen(6) . . . 515
gma-papertokens(6) . . . 517
gma-pfcommon(7) . . . 521
gma-preferences(6) . . . 522
gma-printchar(6) . . . 528
gma-printmon(6) . . . 530
gma-rendersizes(6) . . . 532
gma-roll(6) . . . 543
gma-rotate(6) . . . 544
gma-servercheck(6) . . . 545
gma-session-publish(6) . . . 547
gma-show(6) . . . 555
gma-skillranks(6) . . . 557
gma-spellbook_cost(6) . . . 562
gma-xpsplit(6) . . . 563
gma(6) . . . 566

E Go-GMA Game Program Reference **573**
gma-go-coredb(6) . . . 574
gma-go-image-audit(6) . . . 578
gma-go-map-console(6) . . . 581
gma-go-map-update(6) . . . 589
gma-go-markup(6) . . . 590
gma-go-preset-update(6) . . . 592
gma-go-roll(6) . . . 593
gma-go-server(6) . . . 597
gma-go-session-stats(6) . . . 608
gma-go-upload-presets(6) . . . 611

F Protocol Definitions **613**
form-preamble(7) . . . 614
gma-dice-syntax(7) . . . 621
gma-mapper-protocol(7) . . . 627
gma-markup(7) . . . 690
gma-preamble(7) . . . 695

G How to Install the Mapper on Windows 10 **707**
STEP 1: Obtain and Unpack the Mapper . . . 707
STEP 2: Install Tcl/Tk . . . 709
STEP 3: Configure Mapper for Server Access . . . 714

H How to Install the Mapper on Linux, BSD, and macOS **717**
STEP 1: Install Homebrew (Mac only) . . . 718
STEP 2: Install GMA Mapper . . . 718
STEP 3: Install Tcl/Tk . . . 720

STEP 4: Check for Required Libraries 722
STEP 5: Configure Mapper for Server Access 724

I Player's Handbook for the Mapper **725**
Installing the Mapper Client 726
Configuring the Mapper 726
Starting the Mapper 741
Let's Play! 744
Health Bars 748
Indicating Special Conditions and Movements 750
Gridlines 758
Pointing to Areas of Interest 758
Measuring Distances 758
Distance to All Creatures 760
Chat Messages and Die Rolls 762
Combat Initiative Clock Display 762
Keeping the Current Actor Visible 764
Spell Area of Effect 764
Fractional Grid Snap for Creatures 765
Requesting New Timers 769
Adjusting Your Hit Points 772
Main Menu 775
Have Fun! 775
Hints and Tips 775
Epilogue 777

J Server Quick Reference **781**
Installing the Server 782
Configuration 783
Ongoing Maintenance 787
Setting Up the Web Server 787
Setting up the Web-based Die Roller 788
What to Tell Your Players 788
Using the Server 789

K Cryptographic Signatures **791**

Colophon **793**
Art Credits 793
Dénouement 794

Index **795**

List of Figures

3.1 Party Information Configured into GMA Preferences 18
3.2 Preferences Editor Worlds Tab 19
3.3 Preferences Editor Networks Tab: Add New Network 20
3.4 Initiative Clock Display at Start of Round 2 (left) and After Bob's Held Action (right). 41
3.5 Initiative Clock Display at Round 3, Before and After Skipping. 43

4.1 The Force Drawn Elements to Top Menu Item 51
4.2 Image Placeholder 59
4.3 Simple Dungeon Map Sketch 62
4.4 Bob vs. Goblin 64
4.5 Incremental Reveal of the Map 71

5.1 Original Image Source for Agnes
Art credit: Igor Esaulov 75
5.2 Circular Crop for Token Image 76
5.3 Cropped Image With and Without Background 78
5.4 Example of Generated Mapper File with Images 80
5.5 Finished Character Token On the Map 81
5.6 Stamped Image With Walls Drawn On Them 84
5.7 Dungeon Rooms From Module B2 (GM and Player Versions) . 85
5.8 Splitting the Map Image Into Parts 89
5.9 Creation of Room `B2_58.map` 90
5.10 Adding Concealed Part of Room `B2_58.map` 91
5.11 Concealed Corridor `B2_58c.map` 91
5.12 Room `B2_57c.map` Connected to Room 58. 92
5.13 Room `B2_57c.map` Rooms 59g and 59, Sans Secret Door. 93
5.14 Secret Door From Room 59. 93
5.15 Exploring the Dungeon. 95

6.1 Bare Minimum Character Sheet (page 1) 102
6.2 Bare Minimum Character Sheet (page 2) 103
6.3 Character Sheet Color Schemes 104
6.4 Character Sheet with Highlighted Areas 117
6.5 Player Skill Rank List 119

6.6 GM Skill Rank Matrix 120

7.1 Encounter Run Sheet for GX1, Area A3 126
7.2 Cultures Available for Name Generation 135

8.1 Example progress clock (no steps / one step completed) 161
8.2 Progress Clock Type Editor 163
8.3 Ritual with Some Successes 166
8.4 Ritual with Some Successes and Failures 166
8.5 Requiring Consecutive Saving Throws 168

9.1 Simple GMA Server Configuration 172
9.2 Mapper Display of Combat Turns and Health Bars 178
9.3 Multi-Player GMA Server Configuration 181
9.4 Example `gma go map-console` output 184
9.5 Setting Hidden Condition on a Creature 186

10.1 Chat/Die Roller Tool Window 188
10.2 Some Chat Messages 189

11.1 Epic-Level Multi-Player GMA Server Configuration 210

B.1 Map object class hierarchy 277

I.1 Appearance Tab 727
I.2 Fonts Tab 729
I.3 Font Selection 730
I.4 Dialogs Tab 731
I.5 Die Rolls/Chat Tab 732
I.6 Colors Tab 733
I.7 Clocks Tab 734
I.8 Servers Tab 737
I.9 Servers Tab 738
I.10 Example Health Bars 749
I.11 10-foot spell area of effect. 753
I.12 Indicating that a creature is flying at height of 5 feet. 754
I.13 Setting a condition on a creature. 755
I.14 Example Condition Markers 756
I.15 Mapper With and Without Gridlines. 759
I.16 Using the ruler tool to measure along a path 760
I.17 Chat and Die Rolls Window 762
I.18 Game Clock During Combat 763
I.19 Game Clock Timer Menu 764
I.20 Marking a radius spell area of effect 766
I.21 Marking a cone and line area of effect 767
I.22 Creatures Fitting into Misaligned Spaces 768

J.1 Example Server Startup Script 784
J.2 Example Script to Start All Servers 785
J.3 Example Log Rotation Script 785
J.4 Example Server Initialization File 785
J.5 Example Server Authentication File 786
J.6 Server Configuration for Automatic Updates 787

List of Tables

B.1 Unicode and Markup for Special Characters 300

C.1 Creature Size Codes and Their Attributes 411
C.2 Typography Effects 421

D.1 Full and Shortened Token Image File Names 537
D.2 Output Files For Firefly Token Image. 538
D.3 Token Images and Creature Size Categories 538

F.1 Summary of Client/Server Messages 629
F.2 Summary of Client/Server Messages (continued) 630
F.3 Condition Marker Symbols 662

Preface

It may be strange to see that version 4.*x* or 5.*x* of a software package is described as "alpha" or "pre-release" so that deserves a word of explanation.

The GMA product has been in use for decades, and has been through four major, distinct phases which dramatically changed its architecture and implementation details enough to justify advancing the major revision number. It has been in active use by the author and his friends throughout that time and is stable (although there is a steady stream of new features being added so some parts are works in progress).

With that said, it's not what we'd consider "public ready" quite yet. It is a little too closely tied to our game group's environment and preferences, and thus needs to be generalized a bit, and perhaps some nicer GUI support so the user doesn't need to do quite as much editing of text files, before we feel ready to release it to the world.

Chapter 1

Introduction

> He said he would have a program running in the background to monitor the game. If my true name is ever spoken aloud, the dungeon will smite me even if he's not paying attention. . . . You're not the one being monitored by some kind of auto-smite program.
>
> —Studio Dongo, *Vanishing in a Puff of Logic: The Tediously Charmed Life of RegretMeNo(t) the Elven Wizard*

In the beginning, there was paper. And plastic dice. Also, metal figures. But mostly paper. Lots and lots of paper notes, tick-off sheets, and various implements to help me, a young Game Master (GM)[1] keep track of everything going on in our role-playing games. While there are many things a GM needs to track, the focus of what eventually grew into the GMA automation software was all about timekeeping. Specifically, what time of day is it right now in the game world, and remembering that a number of events are scheduled to happen at various times in the future.

With all of the other things demanding a GM's attention, it is easy to forget about those 5 hit points a creature regenerated a round ago, or the sudden entrance a villain will make 30 seconds from now, let alone the fact

[1] Or Dungeon Master if you prefer, which at that point in history was more accurate since we were playing First and Second Edition AD&D at the time.

that the mysterious artifact a character idly pocketed two sessions ago is going to trigger something unexpected at the next full moon.

And so, out of the usual blend of laziness, impatience, and hubris that Larry Wall tells us are the great virtues of a programmer, I decided that writing some C++ code to automate these tasks was better than my ever-growing pile of scribbled notes to myself.

Thus was born the Game Master's Assistant software, or simply "GMA" for short.[2] In those days, it was a few lines of C++ code and Perl scripts running on my Sun 2 server, with a wonderfully 1990s-esque curses interface.

Timekeeping

The core of its functionality remains the core focus of GMA today, despite numerous rewrites, enhancements, and expansions—keeping track of time. At heart, GMA is a giant clock/calendar. It always knows what time and date it is in the game world. The GM has controls to advance the clock back or forward as needed, or to let it run freely in approximately real-time.

To keep track of what happens during the game, the GMA clock tracks a number of *events*. Originally, these came in two flavors—*timers* and *counters*.

Timers are fairly straightforward. They simply serve as reminders that at a certain time, an event will happen (the timer will "expire" as it were, like an alarm clock going off). Timers may be specified in absolute ("at 3:00 in the afternoon on next Thursday, the old wizard will knock on your door") or relative time ("in 4 rounds, the goblin chief will attack").

Counters keep track of a quantity of something. They came about initially when I had a psionicist PC in our 2E AD&D game who had a pool of Psionic Strength Points. Using her powers required her to spend some of her points, but these would also gradually "regenerate" back to her maximum number of PSPs at a rate of some number of points per hour. Since there was this regeneration aspect to PSPs it seemed convenient to let the computer clock figure out how many points she had at any given time.

The counter tracks the current number of points and the maximum they can reach (when the pool is fully recharged) as well as the regeneration rate. As the clock advances, the counter automatically adds back the points at the given rate. This regeneration can also be turned on or off (since there are situations where the character's point recovery won't function).

Another kind of event, a *drain*, may be attached to a counter. This automatically drains away points from the counter at a specified rate. Like the counter's built-in regeneration feature, drains may be turned on or off at will as well.

At one point I tried to use counters to track hit points, but it proved too cumbersome to set up all the counters before each combat encounter,

[2] At some early points it may have also stood for Game Master's Apprentice.

but it did help for creatures with fast regeneration abilities. The current version of GMA has a new kind of event, a *health tracker* which handles this much more easily, allowing the GM to track creature hit points and other combat related conditions on the GMA console. It also keeps the battle map program updated with these stats and tracks when each creature may act in the initiative round.

A health tracker is like an enhanced kind of counter, so it may also have a drain attached to it. One typical example of this is when a creature is taking bleed damage or is dying; the drain will automatically keep draining hit points due to these ongoing wounds without further action on the part of the GM.

There is also a *condition* event which is like a timer but is associated with a creature in play. It sets a condition (such as prone or unconscious) on the creature and then automatically removes it once its timer expires. It can also be configured to *add* a condition when the timer expires, to automate delayed-onset conditions.

Maps

Beyond these time-keeping duties, the GMA software suite includes support for tactical battle maps of the dungeon or other environment where the characters are exploring. Originally this was a simple Perl script which generated miniature-scaled floorplan maps for use with metal figures.

This has now evolved into a multi-user battle map tool which provides users with a tactical display of the player characters' immediate surroundings and where their opponents are. The players may move their own tokens around on the map. The GM may manually (or automatically via events tracked by GMA) update the maps to show the arrival of new monsters, indicate whose turn it is, and other changes of the conditions in the battle scene.

Since these mapper tools are connected in a multi-user environment, they also provide a player chat channel (including the ability for private conversations) and an online die roller which ensures fair and visible rolls as well as "blind" rolls which only the GM is able to see.

Creatures

GMA includes character and monster definition capabilities sufficient to create and update character record sheets for players and encounter sheets for the GM. The latter provides the necessary stat blocks for creatures for quick reference during combat. The data from those sheets also feeds directly into the GMA console for instant integration into the initiative tracker, creature health trackers, and the battle map clients, as well as setting up one-click die rolls for the GM to roll attack and damage for each creature.

An interactive character sheet tool has been started which facilitates to track character stats in real time. This is still a work in progress but has not yet been completed.

Game System Support

GMA has been used, at various times, to help the author run his AD&D 1st and 2nd edition games, then—along with significant feature upgrades—for D&D 3rd and 3.5 editions. Its current form is designed for Pathfinder (first edition) and is employed for our Pathfinder games. The software is definitely set up for Pathfinder out of the box, but can be adapted (with some programming) to other systems, although it would most easily adapted to d20-compatible games.

API

The GMA system is built on a common framework of library modules which allows custom programs to be added which can interact with the existing toolset. This is implemented in Python, with the exception of the mapper program. The mapper is currently the only remaining part of GMA which is still implemented in Tcl/Tk. It will eventually be ported to the new Python core, at which point the last of the old Tcl implementation will have been retired.

A side project is also underway to port the core API to the Go language. The code is hosted on github at MadScienceZone/go-gma. The API documentation may be viewed at pkg.go.dev/github.com/MadScienceZone/go-gma/v5.

How to Read This Document

This manual is written to give a step-by-step tutorial introduction to the GMA software tools. If you start at the first chapter and read straight through, we will guide you in learning what you can do immediately "out of the box," then proceed to add more and more infrastructure and configuration around the base tools to gain access to more advanced features such as fancy graphics and multi-user networked operation. Each chapter builds on what was learned in the previous ones.

We won't go into excruciating detail about every single feature in the chapter-by-chapter tutorial. That is designed to get new users started and to get to a point where the reader is reasonably familiar with how GMA works and what can be accomplished with it.

In the appendices, you will find a reference section which gives the full technical details. Having completed the tutorial chapters will help you know what to look for, and how to understand, what you find in the reference section.

Appendix A describes the operation of command-line utilities which are not directly part of GMA *per se* but are of general applicability which may be of use when creating automation around GMA.

Appendix B documents the API functions available in the GMA core library. Programmers who wish to extend GMA by making add-on programs that work with GMA may find these functions useful.

Appendix C describes the format of the various data files used by GMA. These are of use to programmers who wish to create their own programs that create or manipulate these data files.

Appendix D documents the operation of the command-line programs which comprise GMA itself, such as the GM console and the mapper client.

Appendix E documents the operation of the command-line programs from the Go-GMA package, including the map server.

Appendix F contains the specification that defines the client/server protocol used by GMA.

Appendices A–F correspond to the manual page entries (in sections 1, 3, 5, 6, (also) 6, and 7, respectively) that are installed with GMA.

Appendix G provides a step-by-step guide to install the GMA mapper client on Windows 10 systems, while Appendix H does the same for Linux, FreeBSD, and macOS. These are intended to be guides a GM can hand out to players who wish to run that program on a Windows or macOS computer. (The rest of the GMA tool set will *probably* work on Windows as well, but this has not yet been extensively tested and is not guaranteed. The rest of this manual assumes a POSIX system such as Linux, FreeBSD, or macOS.)

Appendix I provides a player-centered tutorial to installing and using the mapper client. (Windows users should follow Appendix G and then proceed to Appendix I, while everyone else should follow Appendix H and then Appendix I.)

Appendix J is a quick-reference guide for setting up a GMA server.

Appendix K has a copy of the public key used to sign the GMA software releases. If you have a need to verify a signed software package, you can copy the public key here for that purpose.

Typographical Conventions

We will use *italics* for emphasis and to introduce key terms. Stronger emphasis will be indicated by using **boldface** type.

Text which has been changed recently is set in blue type.

Terms which are entered literally as shown are set in `typewriter font`, and values which indicate "variables" or "placeholders" which you'll need to substitute with the actual value needed are set in ⟨*bracketed italics*⟩. (The brackets are not typed, but serve to distinctly show where the variable text is. They are designed to look differently in print than literal angle brackets in the typewriter font, which look like this: <>.) Optional items of code are enclosed in [square brackets] in the normal running text font.

Menu selections are indicated as File » Show » Something which means to click on the "File" menu in the application's menu bar, then select the "Show" submenu, and finally the "Something" selection from that menu.

```
Excerpts from data files are shown in typewriter font in a bo
xed frame, with hard line wraps at the end of the box.
```

```
Examples of command-line input and terminal output are shown
in the same way, but in a shaded box and with a $ prompt
before each command.
$ like this!
```

The notation `$GMA` is used to refer to the directory where GMA is installed on your system, and by extension `$GMA/bin` is the directory where the executable programs are installed, `$GMA/lib` is the start of the directory tree where the GMA API library modules live. If you install GMA in `/opt/gma`, then `$GMA` would mean `/opt/gma`, `$GMA/bin` refers to `/opt/gma/bin`, and so forth.[3]

[3]The notation comes from standard Unix shell variable notation. In fact if you define a shell variable `GMA` appropriately (e.g., `GMA=/opt/gma`), then you can literally use `$GMA` in commands and it will work.

CHAPTER 2

Installation and Setup

> For a lot of people, "Dungeons & Dragons" has been a hard thing to describe. I can't tell you how many social environments I've been in where I say, "I play D&D," and a bunch of normies will be like, "How does the game even work? What's that like?" I didn't have anything to really describe it that didn't make me sound like a crazy person.
>
> —Matthew Mercer

Before we can get started running games with GMA, it is (of course) necessary to have GMA installed. We will describe the process to do that here. Since GMA has relatively few dependencies, this should be a painless process on most systems.

Platform

The GMA software has been developed and used on some sort of Unix-derived operating system throughout its lifetime, starting with System V Unix, then SunOS, and eventually including Linux, FreeBSD, and macOS. If you're using one of those platforms or something similar, it should be a fairly simple matter to get GMA running on your system.

If you wish to use something substantially different (say, VMS or Microsoft Windows), then it will *probably* work fine, since GMA is written in portable languages (primarily Python and Go, with some legacy code that's still in Tcl, Perl, and shell scripts). However, this has not been extensively tested so it's also likely that not *everything* will work as it should, and may require some tweaking to sort out. (Although see Appendix G starting on page 707 for instructions to install and run the mapper client on Windows 10, and Appendix H on page 717 for the other supported platforms.)

For that reason, the rest of this guide will assume a Unix-like system. Adventurous souls who insist on pushing forward to use GMA on Windows (or their old Amiga system or whatever) are thus on their own. We wish you the best of luck on your quest.

Pre-release Implications

You'll notice that we have a chapter devoted to installing GMA rather than just saying to run a command like "`pkg install gma`" which will do all the magic for you. This is because GMA is currently under active development and is **not released publicly** yet. We'll think about fancy packaging when we're closer to feeling ready to let the general public have access to GMA.

In the mean time, if you're reading this and installing GMA, then you either belong to our role-playing group, agreed to help test and develop GMA, got permission from the author to preview/use GMA, or acquired a bootleg copy of GMA to carry out some diabolical plan of your own (and you should be ashamed of yourself).

This also means that you should not expect a great deal of stability from GMA. We're still working on it.

(The information in the above paragraphs is becoming increasingly incorrect as we release more and more of GMA as open source software. As of the time of this writing, only the GMACore code remains unreleased, and we're working to move it closer to a state where we can release it as well.)

Prerequisites

GMA has few dependencies, and many of those it does have should be installed already on a typical system. Make sure these are on your system already.

Git

The pre-release GMA code is obtained by cloning its git repository. Install Git on your system if you don't already have it ready. Generally this is available from your operating system's app store or package management system. For example, FreeBSD users can run "`pkg install git`" or Linux

users can similarly use `apt` or `yum` or similar program to install the `git` package.

Python

Likewise, your platform should give you access to install a Python interpreter (and probably already has it installed). Be sure to get the latest version of Python (we require at least version 3.8). Be sure it supports the latest version of `tkinter`. If you're on a macOS system using the stock Python interpreter supplied with the OS or are using homebrew, you may need to manually compile or update Tcl and/or Python to be sure you have a fully working Tk runtime. The Mapper installation instructions in Appendix H are a good starting point for setting up Tcl/Tk on a Mac; you should do that first, then manually install Python with build options to use your just-installed Tcl/Tk instance for Python's `tkinter` package.

Dependencies

The GMA software depends on the `tkinter-tooltip` package which does not come standard with `tkinter`. It may be installed using `pip` or whatever equivalent Python package manager you use on your system. For your convenience, a copy of this package's source distribution is included in the `dependencies` directory in the GMA repository.

Tcl/Tk

Until the mapper client is ported to the new Python or Go code base,[1] you'll need to have a Tcl interpreter installed to run that program. Install the latest version of Tcl and Tk on your system (at a minimum, version 8.6 is needed).

You'll also need to make sure `tcllib` and `tklib` are installed. An easy way to do this is to install the GMA Mapper program and follow the instructions on page 713 (Windows) or page 722 (other platforms).

Perl

Perl, like Python, is likely already installed on your system. You'll need a fairly modern version of Perl for a couple of the supporting tools used to prepare games by the GM. Specifically, it is needed by the `rendersizes` tool which prepares images for use in the mapper, and if you are *very* old-school, the `mkmap` script which prints out large paper maps for use with miniature figures.

[1] Assuming that ever happens.

Go

Having a Go compiler on hand is only necessary if you are going to use the Go-GMA package, but since the current implementation of the GMA server is in that package, if you plan to use GMA in its full multi-user form, you're going to want that. Go version 1.18 or later is required.

SSH and SCP

If you want to use the advanced server features described in Chapter 11, you will need the `ssh` and `scp` programs installed and will also need to have a personal SSH certificate to authenticate you to the server. You will also need an SSH certificate to authenticate to the GMA Git repository, as we'll see in a moment.

Curl

To use the `mapper` client with image tiles, you'll need the `curl` program (which is used by the mapper to do the actual file retrieval from the server). This should be easily obtained from the same package repostory your platform offers as the other programs mentioned above.

PostScript

Paper reports generated by GMA, such as character record sheets, or encounter run sheets, are generated as PostScript documents. You will need a way to view or print PostScript files to paper, or a program such as Ghostscript or `ps2pdf` to convert them to PDF files.

ImageMagick

If you are preparing images for use with the mapper, you will also need the ImageMagick application installed. This is employed to make a number of image manipulations as scripted by `rendersizes`.

Installing GMA

Decide where you want to install the GMA software. In its pre-release state, it is best to simply run it out of the source directories rather than installing it into system areas such as `/usr/local/bin`. For personal use, you could just install them into a directory such as `~/gma`, or for wider use you may choose a location such as `/opt/gma`.

Go to the directory where you want the `gma` directory to be placed in, and clone the repository:

```
$ cd /opt/games
$ git clone git@github.com:MadScienceZone/gma.git
Cloning into 'gma'...
remote: Enumerating objects: 9023, done.
remote: Counting objects: 100% (9023/9023), done.
remote: Compressing objects: 100% (5726/5726), done.
remote: Total 9023 (delta 5542), reused 4419 (delta 2595)
Receiving objects: 100% (9023/9023), 50.96 MiB | 5.72 MiB/s,
done.
Resolving deltas: 100% (5542/5542), done.
Updating files: 100% (854/854), done.
$ cd gma
$ export GMA=/opt/games/gma
```

(If you don't use SSH to access github, you may need to use the alternative command

```
$ git clone https://github.com/MadScienceZone/gma.git
```

This will place a `gma` directory containing the GMA code in your current directory. That last line ("`export GMA...`") creates an environment variable called `GMA` which we will use in this manual to make it convenient to refer to the directory where GMA is installed. You can make this variable permanent in your environment to continue using it every time you run GMA.

Installing GMA Mapper

You'll also want the mapper client to be installed. Since we just used Git to clone the GMA Core repository, we'll just do the same for the mapper.

```
$ cd /opt/games
$ git clone git@github.com:MadScienceZone/gma-mapper.git
```

(As above, use the `https:` version instead if you don't use SSH for github.)

Installing Go-GMA

While you don't necessarily need Go-GMA, if you intend to run GMA in its full multi-user mode, you'll need the server program from Go-GMA.

```
$ cd /opt/games
$ git clone git@github.com:MadScienceZone/go-gma.git
```

(Again, as above, use the `https:` version instead if you don't use SSH for github.)

Environment Variables

Set the `GMA` environment variable to the name of the GMA top-level directory (as shown in the previous example above).

Add `$GMA/bin` to your `$PATH` environment variable so that the commands there may be executed at the shell prompt without further ado.

If you want to specify a specific interpreter for Tcl/Tk, Python, and/or Perl scripts, set the `$GMA_WISH`, `$GMA_PYTHON`, and `$GMA_PERL` variables accordingly. For example, you might put the following in your `~/.profile` file or equivalent (using the correct pathnames for where you installed GMA and GMA Mapper, and where these programs live on your system):

```
export GMA=/opt/games/gma
export PATH="$PATH:$GMA/bin"
export GMA_MAPPER=/opt/games/gma-mapper
export GMA_WISH=/usr/local/bin/wish8.6
export GMA_PYTHON=/usr/bin/python3
export GMA_PERL=/usr/local/bin/perl
export GMA_IGNORE_GO_GMA=1
```

If you have Go-GMA installed as well, do not set `$GMA_IGNORE_GO_GMA` but instead, assuming you have installed it in `/opt/games/go-gma`, set

```
export GMA_GO_GMA=/opt/games/go-gma
```

Adding the GMA `bin` directory to your `$PATH` is the most important one. Setting `GMA` is a good idea (but GMA will try to figure this out if you don't). The other three may not be necessary for you unless your particular system doesn't put those interpreters in a standard location or you need a specific version of those tools for GMA.

See the documentation for the `gma` program starting on page 566 for a complete description of the environment variables used by GMA and what you'll need to set for your particular installation.

At this point, you have GMA installed. No further setup or configuration is required to just dive in and get started. Coincidentally, that is the topic of the next chapter. As we progress from chapter to chapter, we'll gradually build up a more complete configuration of GMA along the way.

CHAPTER 3

UNBOXING: DIVING RIGHT IN

> One of the things stressed in the original game of D&D was the importance of recording game time with respect to each and every player character in a campaign. In AD&D it is emphasized even more: YOU CAN NOT HAVE A MEANINGFUL CAMPAIGN IF STRICT TIME RECORDS ARE NOT KEPT.
>
> —Gary Gygax
> "Time in the Campaign", *AD&D Dungeon Masters Guide*

WE UNDERSTAND THE APPEAL of taking a new toy right out of the box and playing with it immediately without taking a lot of time to read the instructions or go through some kind of lengthy assembly or configuration process. We're impatient like that, too.

So let's just get going with GMA.

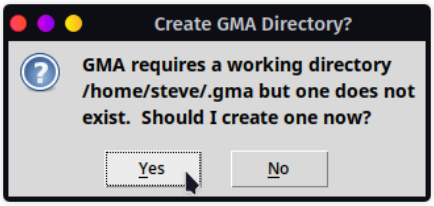
This chapter doesn't assume you've done anything at all to set up GMA beyond installing it from the Git repository.

```
$ gma
```

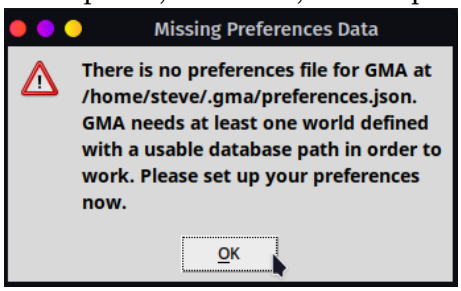
Note that this requires the `$GMA/bin` directory to be in your `$PATH` in order to work.

This launches the main GMA tool, often referred to as the GM console. (This is a Python script. If for some reason it won't run on your system, it may be due to some issue finding the appropriate Python interpreter for it. Set the `$GMA_PYTHON` environment variable to the path to the correct Python interpreter for your system in that case.)

The very first time you run `gma`, it will prompt you to create the `~/.gma` directory:



Say "yes." All of the GMA tools place their working files in this directory, including configuration options, databases, and temporary image caches.



The GMA programs use a common preferences file, by default stored in the `~/.gma/preferences.json` file.[1] Since we're just setting up a new installation, that file doesn't exist yet, so GMA can't continue until we set one up.

When we click the OK button, GMA will open a preferences editor window.

Configuring GMA Preferences

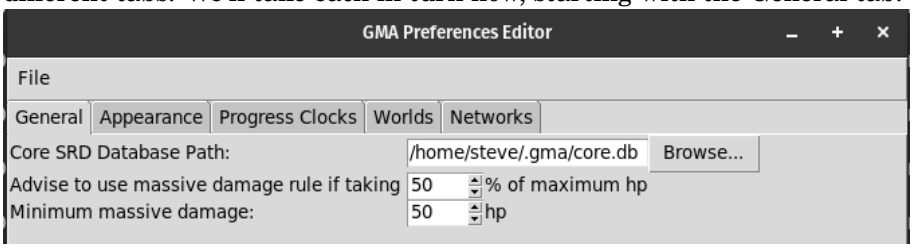
Now we need to set up the preferences file. If you're running GMA for the first time, it will open its preferences editor to set up the initial file. Subsequently, you can make changes by selecting Edit » Preferences... from the main menu.

If you're not running a GMA program and want to make changes to the preferences directly, the easiest way to do this is to run the stand-alone preferences editor.

[1] You can use a different file if you wish, specifying it with the `--preferences` option. You can even maintain a set of different preference files for different purposes, although that isn't often necessary since a single file has the ability to hold preferences for multiple campaigns and servers.

```
$ gma preferences
```

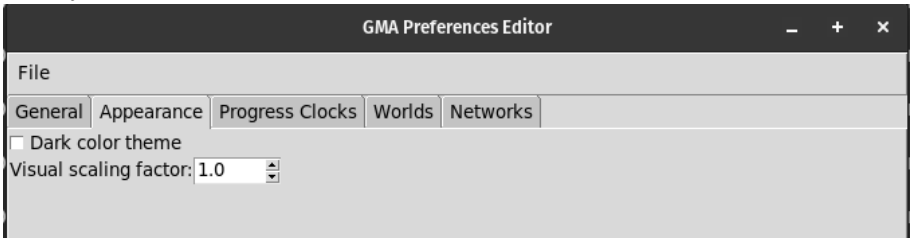
In the preferences editor, the various settings are organized into four different tabs. We'll take each in turn now, starting with the General tab.



Versions of GMA prior to 7.0 don't use the Core SRD Database Path, so we'll ignore that for now.

If you're playing with the optional death from massive damage rule, configure the appropriate values for it here. It defaults to the Pathfinder 1st edition settings where a single attack that deals 50% of a creature's maximum hp (minimum 50 hp) triggers the rule. If this happens, GMA will pop up an alert box to let you know to call for the appropriate saving throw.

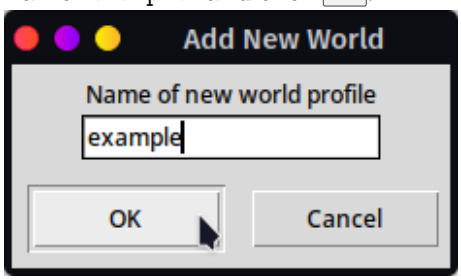
If you don't use that rule, set these values to 0 (zero).



If you want to use a dark color theme, check the box. For most systems, a scaling factor of 1.0 (the default), is right. However, if you are on a high-resolution display and the widgets look too small, try increasing this number to 2.0 or larger.

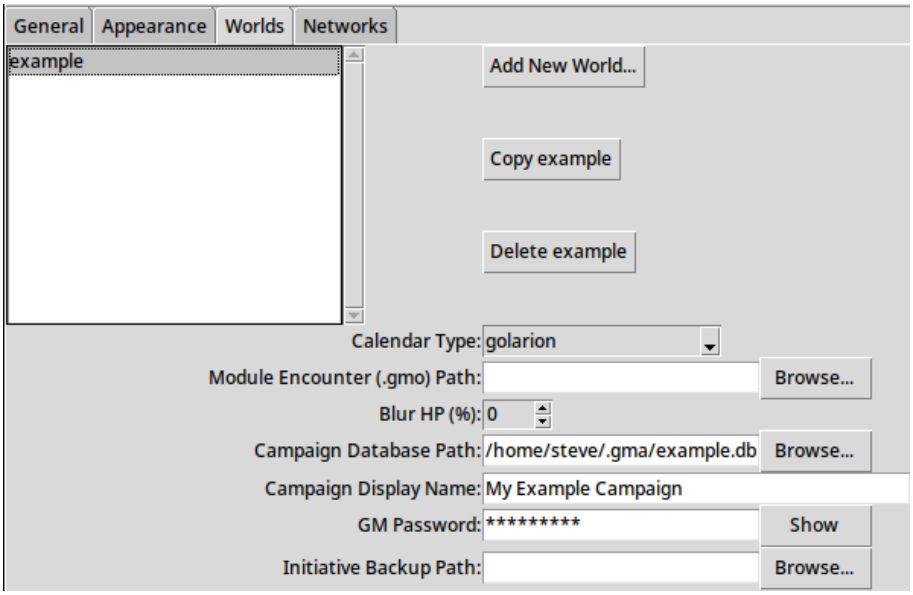
The Worlds tab contains the settings that are specific to a particular campaign. You can configure any number of campaign worlds here, and easily switch between them as needed. To create a new one, click on the Add New World... button, then enter the name for the campaign.

We'll enter the name "`example`" and click OK.



This adds “example” to the list of worlds. With that world selected in the list (by clicking the mouse on it), the settings that go along with that world appear in the rest of this tab and can be edited.

If another world is selected, then that world’s settings can be edited. Whichever world is selected when the preferences file is saved, that will be the world GMA will use when it starts operations.



We’ll continue filling out the various settings for the example world profile we are creating now.

The calendar type selects from among the different calendar systems GMA supports. We’ll choose the Golarion calendar.

We can leave the Module Encounter and Blur HP values empty for now. We’ll learn about this them when we get to Chapter 7 (page 123).

The Campaign Database Path is the full pathname for the database GMA should use to track time, events, and other information about the campaign in play. Each campaign world must use its own database file. If the file does not yet exist—as is the case here—just type the name of the file you want to use and GMA will create it when you start it again. If the file already exists, you can click the Browse... button to navigate to the file.

The Campaign Display Name is the name of your game campaign as you wish it to be displayed by the various GMA programs.

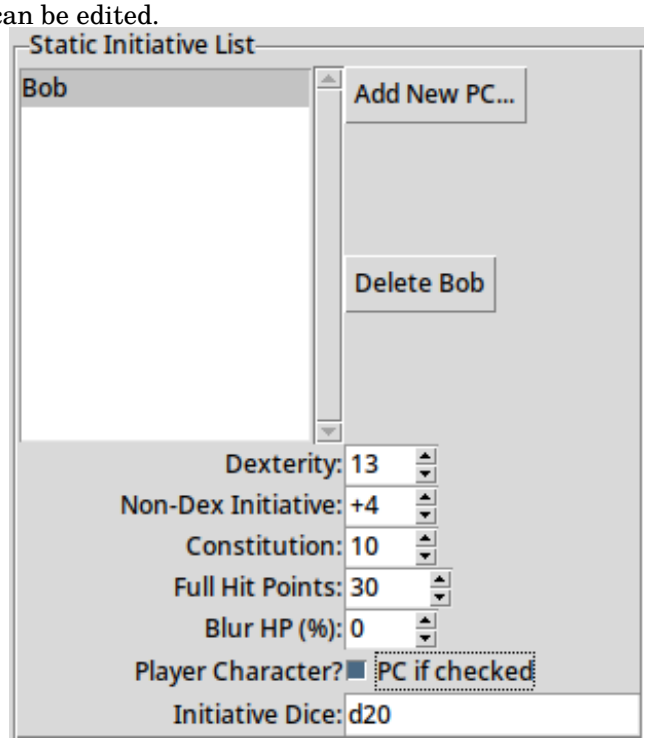
If you’re using a server, enter the GM’s access password here. You can leave this empty for now, until we set up the server in Chapter 9 (page 171).

You can also leave the Initiative Backup Path empty until we start using encounter files and running real combat.

Entering Party Information

Next we need to enter information about the player character party in the bottom portion of the tab. In the box labelled "Static Initiative List" click on the Add New PC... button and type one of the characters' name. Let's enter Bob. This adds Bob's name to the list of people who will always be loaded into the initiative list whenever we go into combat.

Just like the list of worlds, when we click on one of the party names, such as Bob here, the remaining fields in this area which are associated with Bob can be edited.



We go ahead and enter Bob's Dexterity score, any additional modifiers to his initiative other than from Dexterity, his Constitution, maximum hit points, and whether to "blur" the display of his hit points. The latter of these is a way to partially obscure the exact hit point total when rendering health bars on the map. Normally, GMs will use the general Blur HP control up above to make all creature hit point reports approximate, but will set the Blur HP control down here for the PCs to 0 so that they show accurately.

Sice Bob is a PC, we check the box to indicate that status. The Initiative Dice value can be left blank to default to using a d20 when rolling initiative for Bob. If needed, a special value can be entered here. For example, if Bob can roll twice and use the better of the two results, then enter "`d20 best of 2`" in the Iniative Dice box.

We'll continue to do the same to enter the rest of our party.

Now we turn to the other box at the bottom of the tab, "Static Caster

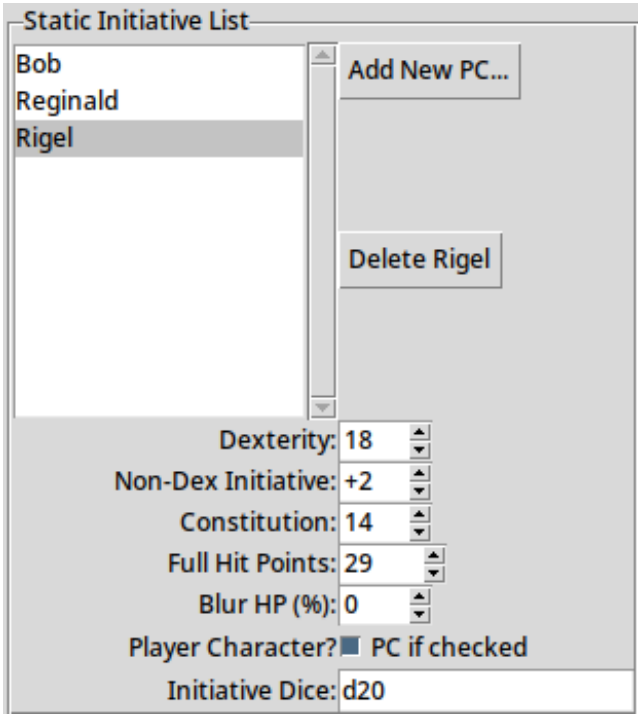


Figure 3.1: Party Information Configured into GMA Preferences

List" where we enter information about the spellcasters. Bob can cast spells, so we click Add New Caster..., type Bob's name, then fill in the Caster Level value of 3 since Bob is a third-level caster.

(See p. 118 to see how this party information may be imported directly into GMA from the character record files in either GMA's `.gcr` format or Hero Lab's `.por` files.)

With that done, we can move on to the Networks tab. This works like the Worlds tab, but it manages all the network connections you need for the servers you will connect to while using GMA.

Start by clicking Add New Network... and enter "`offline`" as the network name (Fig. 3.3).

Leave the hostname field blank. When this network is selected, GMA will run in offline mode, not attempting to contact a server at all.

When we have a server to connect to, we can add another entry for it. For example, we can add one called "`example server`" with hostname "`example.com`" and an appropriate port number.

Whichever network profile is selected when the file is saved, that will be the network GMA will use when it starts each time.

When finished, select File > Save Preferences (to default file) from the menu.

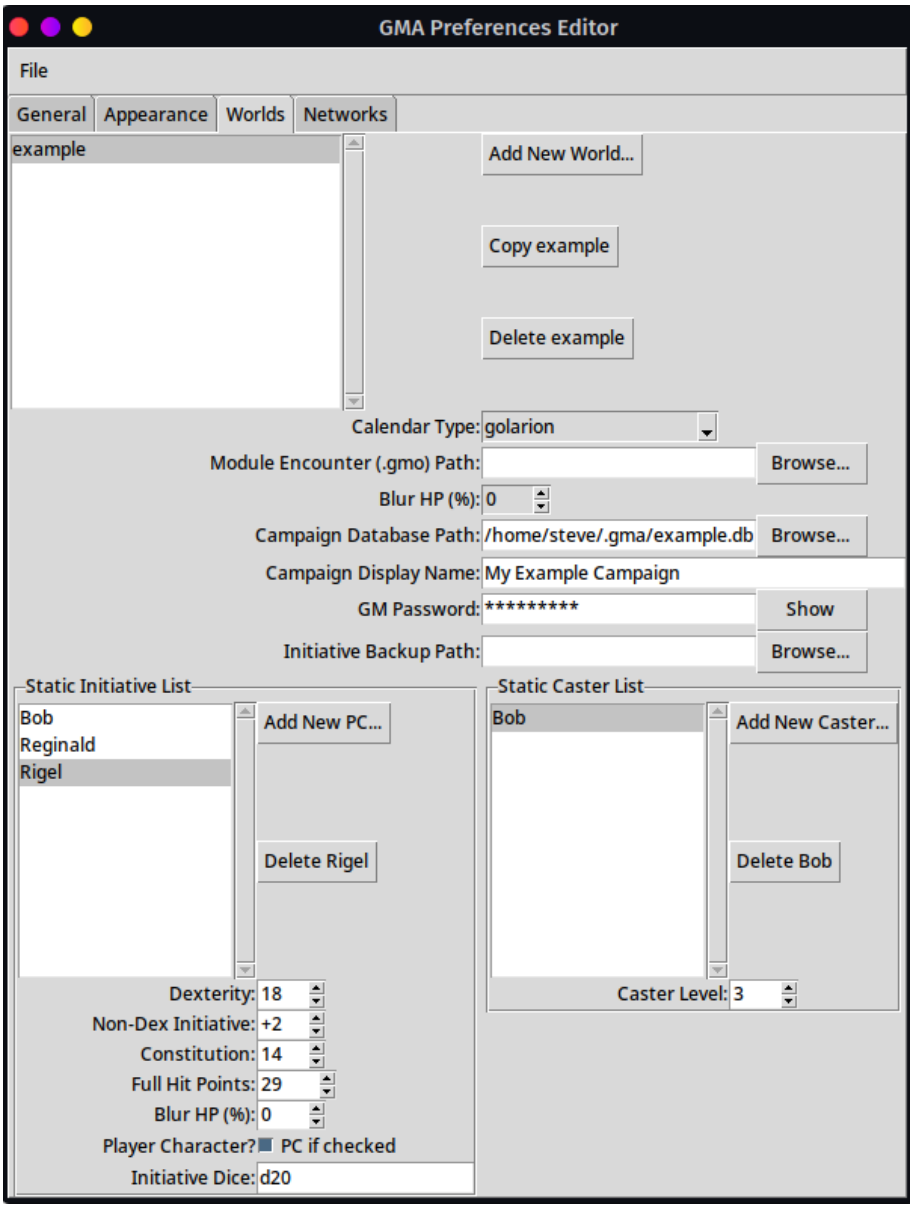


Figure 3.2: Preferences Editor Worlds Tab

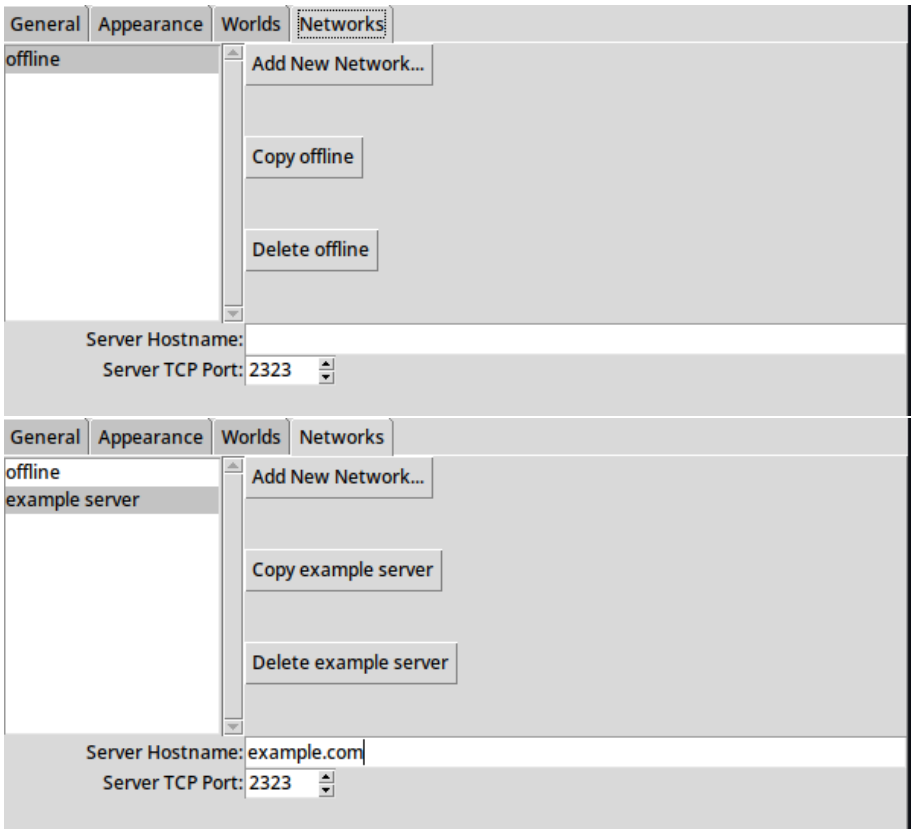


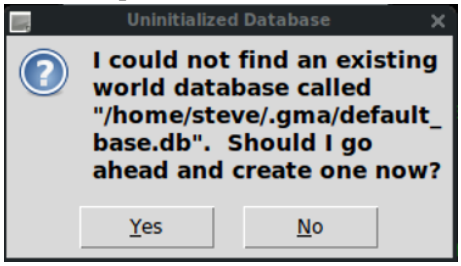
Figure 3.3: Preferences Editor Networks Tab: Add New Network

Then exit the preferences editor and start GMA again.

See the reference information for `gma preferences`(6) on page 522 for details about configuring GMA.

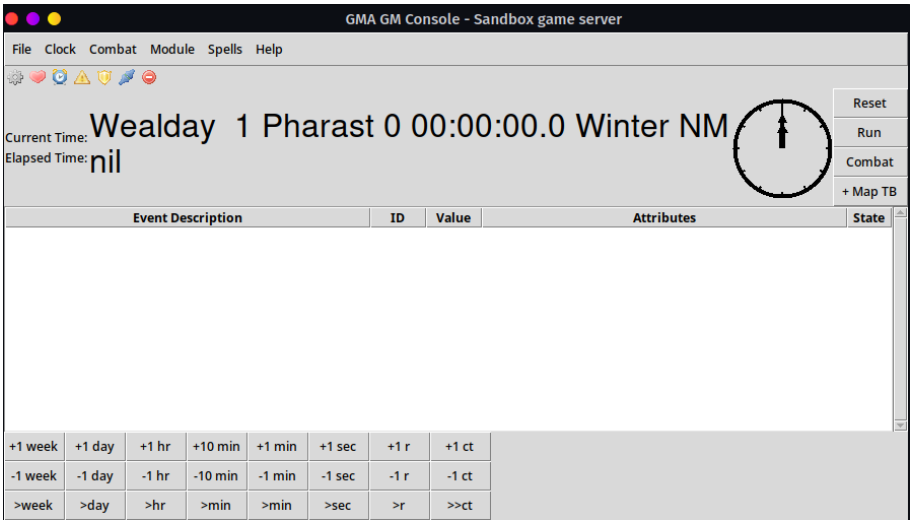
Back to GMA

Once we are done setting up the initial preferences, GMA will resume preparing everything for it's first operation.



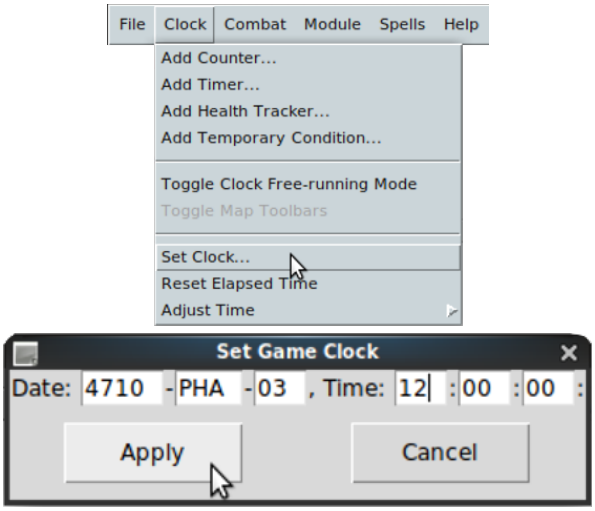
You've probably guessed by now that it's best to just let GMA set things up as it wants, so say "yes" to this as well. Each world includes a database where GMA will track the events and other information it's managing for that world. GMA is setting up a database now for our default world.

At last, GMA is ready for use and we see the main screen:



As you can see from the rather heavy emphasis on time in this main window, the main focus of the GMA tool is keeping track of time and timed events within the game. By default, it uses the Golarion calendar from the Pathfinder game, but others are supported as well. For now, let's just use what we have. We'll want to start by setting the clock to the current date and time. Let's say it's noon on the 3rd day of Pharast in the year 4710.

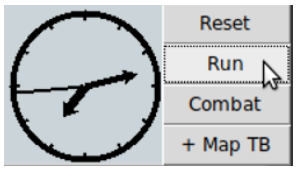
Select Clock » Set Clock... from the menu and enter the current date and time.



Now you can keep track of the passage of time in the game by clicking the buttons along the bottom of the main window. These advance the clock forward, or move it back, in increments of weeks, days, hours, 10 minutes, minutes, seconds, rounds, and initiative counts.

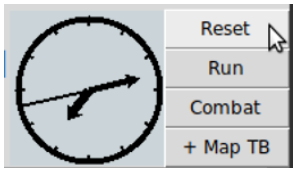
An initiative count is 1/10 of a second on the clock. When in combat mode, where we are interested in counting time in terms of rounds and when each participant can take an action, the initiative count is the time increment represented by a single initiative "slot." Thus, a combat round (which is 6 seconds of actual time) is divided into 60 initiative counts, allowing for 60 "slots" in which up to 60 individuals can be assigned a time to act.

Choosing Clock » Toggle Clock Free-running Mode or clicking the Run button to the right of the clock face starts the clock's free-running mode. This may be convenient to provide a visible indicator to the players (assuming the clock display is projected where they can see it) that the game world's time is moving forward and their actions are "live" as it were. Clicking the button again stops the clock.



In addition to the absolute time clock that displays the date and time in the game world, there is also a relative time clock the GM can use to measure the time since an interesting point of reference. This is like a stopwatch that is constantly counting forward but which you can reset to zero to mark an event you want to note the elapsed time since it occurred. The most common use for this is to track the time elapsed since the beginning of a combat encounter, as we'll see later.

To reset this elapsed counter, press the Reset button to the right of the clock face, or choose Clock » Reset Elapsed Time from the menu. Along the bottom row are a set of buttons which, rather than simply moving the clock forward or back by the given time increment, move the clock forward to the start of the next such increment. Note that these are based on the *relative (elapsed) time* counter, not the current time of day. Thus, clicking the >hr button advances the clock to the start of the next full hour since the elapsed time clock was reset.



These functions are also available from the Clock » Adjust Time menu.

There are two special cases. The >week button advances the elapsed time clock to the start of the next day on that clock, that is also on the first day of the week on the current time clock. The >>ct button advances the clock to the next initiative count in which there is an active combatant scheduled to act.

To advance to the start of a time increment on the actual time-of-day clock, hold the shift key (⇧) while clicking on the >hr, >min, etc., but-

tons. For example, if the current time of day is 11:34:12, holding ⇧ while clicking >hr will result in the time being 12:00:00.

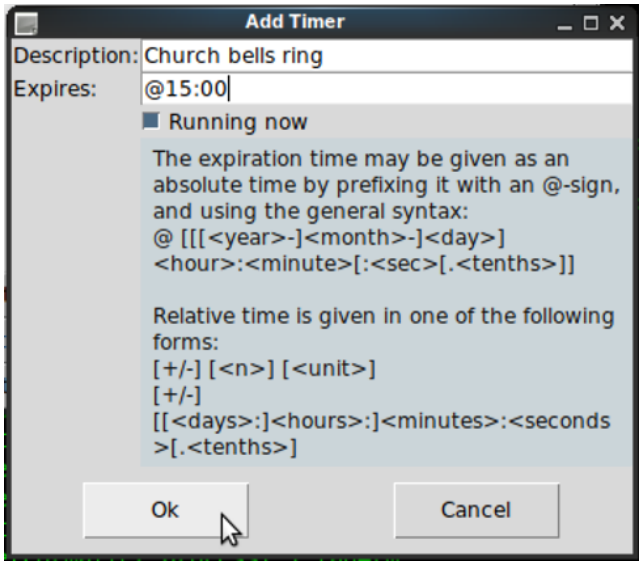
Scheduled Events (Timers)

Keeping track of what time it is, and how long has elapsed since the goblin chieftain surprised the party, is all well and good, but during a tabletop game session there are many specific events that the GM needs to track. GMA's timers come to the rescue here.

A *timer* is like setting an "alarm clock" on the game clock—at a certain time, it will *expire*, indicating that whatever event it represents in the game has happened.

Let's say that at 3:00 the church bells will ring out to announce the start of the town festival. The GM wants to be aware of when that exact moment comes up so she can inform the players that they hear the bells peal even if they're in the middle of a big action scene. To avoid forgetting or missing the timing, the GM enters a timer into GMA.

Select Clock 〉 Add Timer... from the menu, or press the timer button on the toolbar (). This brings up a dialog to enter the parameters of the timer you wish to set.

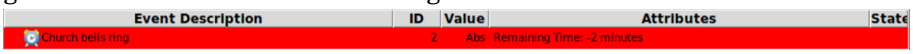


We enter a description that tells us what the event means, and since we want to schedule it to occur "at" a specific time of day, we start the expiration time with an at-sign (@). In this case, we enter "@15:00" to have the event trigger at 15:00 (i.e., 3:00 PM) today.

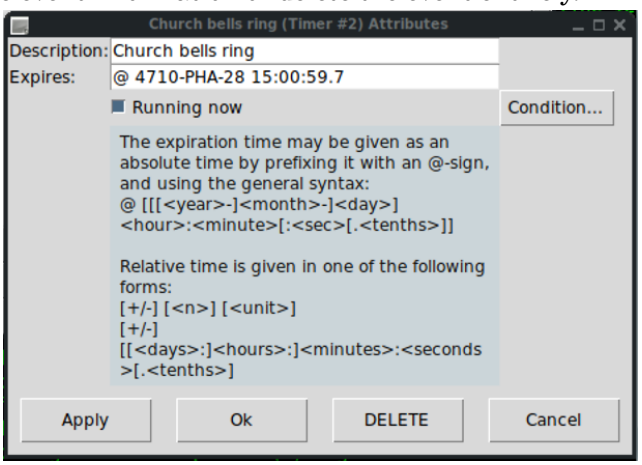
When we click Ok, we see that the event has been added to our main display.



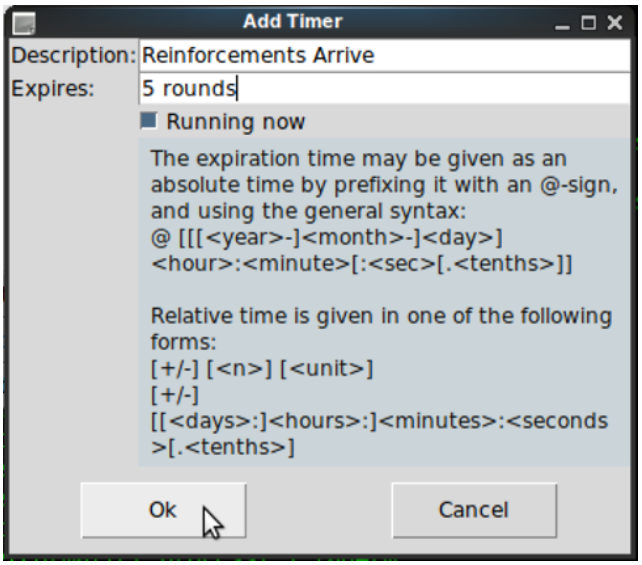
As the time of that event draws closer, the indicator in the event list will show how long still remains until it happens. Once it expires, the background for that timer turns red to get the GM's attention.



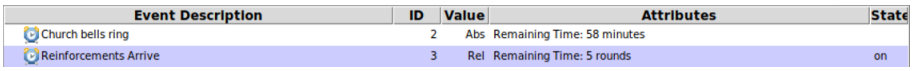
If we click on the right half of the event (in the "Attributes" section), we can edit the event information or delete the event entirely.



Sometimes we want to note a future event that happens in a *relative* time interval rather than an *absolute* time of day. For example, let's say that in 5 rounds some reinforcements will arrive on the scene. Again, we create a timer event but this time we specify the length of time until the event happens:

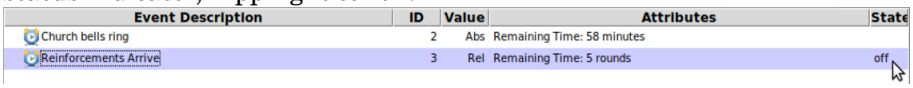


Now we have two events coming up. The alternating background colors are only to help distinguish the rows on the display.



Event Description	ID	Value	Attributes	State
Church bells ring	2	Abs	Remaining Time: 58 minutes	
Reinforcements Arrive	3	Rel	Remaining Time: 5 rounds	on

If something were to delay the reinforcements, we can click on the "on" status indicator, flipping it to "off."



Event Description	ID	Value	Attributes	State
Church bells ring	2	Abs	Remaining Time: 58 minutes	
Reinforcements Arrive	3	Rel	Remaining Time: 5 rounds	off

This means that the timer is no longer running. It will remain frozen, not counting down to its expiration, until turned back "on" again. We can also turn it on and off from the event editor dialog by checking or unchecking the "running now" box and then clicking Ok or Apply.

Randomly-Scheduled Timers

Sometimes you need to set an event for a randomly-determined length of time. For example, maybe Reginald the Paladin was hit by a magical effect that stuns him for 1d6 rounds. You could roll a six-sided die and enter the result into GMA when setting up the timer, of course. But you can also let GMA roll the dice for you. If part of the time specification is enclosed in square brackets, such as "`[1d6] rounds`", then GMA will generate a random number (in this case, between 1 and 6) and use that value for the timer's duration.

Generalizing this a bit, in many of the numeric entry fields in the timer dialog box (as well as other data entry forms we are about to discuss), instead of typing a simple number like "10" you can place a simple math expression enclosed in square brackets like "`[8+2]`" and GMA will use the expression's value for that field. The expression may include die roll expressions as well, as explained on p. 621, allowing you to enter something like "`[2d6+3]`" to have GMA randomly determine a value for you.

It is generally important to remember that these expressions are evaluated once and their value used from that point forward. For example, we'll see shortly how we can schedule recurring effects like the "regeneration rate" for counters where we can specify that rate using straightforward values such as "`minute`" or "`round`", or even "`2 rounds`" to have a number of units restored every 2 rounds, but alternatively you could specify "`[1d4] rounds`" to have GMA pick a random number from 1–4 and use that as the number of rounds between each regeneration. But what that actually means is that GMA rolls the 1d4 to get the random number—let's say it gets a 3—and then sets up its internal regeneration timer to regenerate that resource every 3 rounds from that point forward. It does not keep rolling a new 1d4 after every regeneration to come up with a new regeneration time.[2]

In the case of die rolls made with bracketed expressions, it should also be noted that these are performed using GMA's internal die-rolling library.

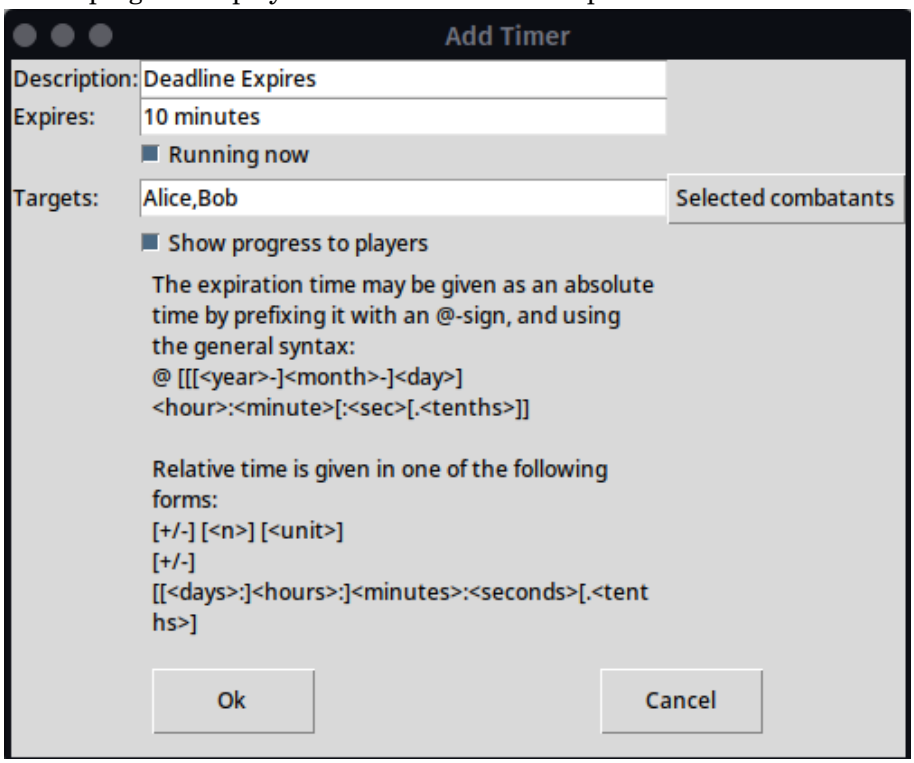
[2]The only thing that supports something along those lines is the progress clock. These are explained in Chapter 8.

If you want instead to use a connected GMA server to handle your die rolls, so you're using the same random number generator as the other rolls in your game or so the rolls are openly displayed to your players, then enclose the expression in double brackets, like so: "`[[2d6+3]]`". If the expression begins with a question mark, as in "`[[?2d6+3]]`", then the fact that dice were rolled "behind the GM screen" as it were is apparent to the players but the result is hidden from their view.[3]

Public Timers

As the GM, you have a view of all timers and can see directly when they'll expire. Your poor players, however, have no such privilege. They can't see any of them. However, perhaps they should see a few selected ones. If there's an event that is important to the players, and their characters would reasonably be expected to be able to keep track of how much time remains before the timer expires, you can choose to make the timer public, so the players can also see its progress.

On the dialog where you create or modify a timer, simply check the "Show progress to players" checkbox to make it public.

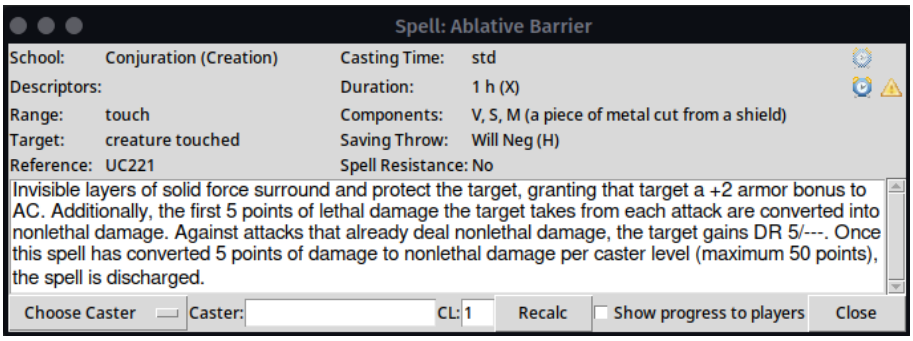


[3] If you use the double-bracket notation but no server could be reached to make the die rolls, GMA will fall back to an internal die-roll operation.

If the timer is associated with one or more players, enter their names in the "Targets" entry box, with each name separated by commas. To make this easier, you can check the boxes for each target character on the initiative display dialog, then click the Selected combatants button to have that list filled in to the "Targets" entry for you.

The players may choose to see all public timers, or only those which contain their name (i.e., the name under which they logged into their map client) in the "Targets" list.

Since this is often used to make the duration of spells visible, you can also create a spell duration timer directly off the spell viewer window. Choose a spell from the menu (e.g., Spells » Spells A–E » A » Ablative Barrier).



This conjures a window with the spell description in it. It also calculates casting time and duration based on the spellcaster's caster level (either by selecting one of the party's spellcasters from the Choose Caster menu button, or entering the NPC caster's name and CL in the entry boxes provided). Now, clicking on the timer button ( ) next to the casting time or duration creates a timer which tracks the time until that happens.

If the "Show progress to players" box is checked, it will be a public timer exposed to the players.

Since casting a spell often imposes a temporary condition on one or more target creatures for the duration of that spell (e.g., they gain the "hasted" or "invisible" condition), you can also click on the temporary condition button ( ) next to the spell's duration. This brings up the same temporary condition dialog as described on p. 44, but it has its values preset to remove the condition when the spell duration expires.

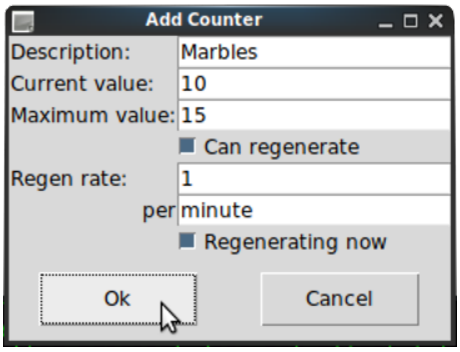
As of this writing, there is no way to designate a set of targets for the regular timers for casting time or duration created from a spell window. This is expected to be corrected in a future release, but for now if you wish to assign targets to the timer, you will need to create it first, then click on it in the GMA event list to edit it. There you can add the targets. However, if you have a caster name selected when you create the timer, that person will be added as a "fixed" target for the spell timer. This means they will be a target of the timer even if you go back and modify the target list later.

Counters

In the next room of the dungeon is a curious device. A bowl of marbles sits under a tiny hole in the ceiling. Once every minute, another marble falls from the hole, landing in the bowl. Once the bowl contains 15 marbles, no more marbles drop.

This somewhat contrived example models a common situation in our games—a pool of points, mana, *ki*, marbles, etc., which may have its number adjusted by player action, and replenishes at some constant rate over time.

To create an event to keep track of this in GMA, we will create a *counter* by choosing Clock » Add Counter... from the menu or clicking the counter button on the toolbar (). This brings up a dialog to enter the information about the new counter:



We give the current number of marbles in the bowl (10), and the maximum capacity it can handle (15). We also check the "Can regenerate" box to indicate that the marbles automatically replenish, and give the regeneration rate of one marble per minute. By checking the "Regenerating now" box, we are saying that GMA should add a marble in a minute, and every minute thereafter, until the maximum value is reached.

Once we click Ok, we see that the counter has been added to our event list:

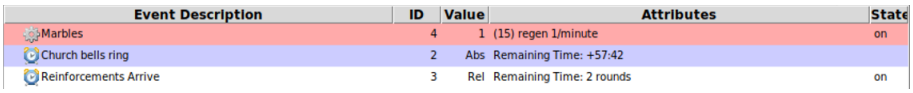
Event Description	ID	Value	Attributes	State
Marbles	4	15	(15) regen 1/minute	on
Church bells ring	2	Abs	Remaining Time: +57:42	
Reinforcements Arrive	3	Rel	Remaining Time: 2 rounds	on

The background for a counter in the event list is the same alternating color pattern used for other events if the counter is at its maximum value.

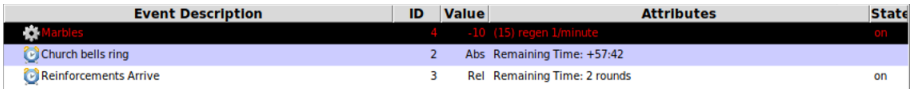
It is green if it has most of its value, but turns yellow and then red as they are depleted.

Event Description	ID	Value	Attributes	State
Marbles	4	10	(15) regen 1/minute	on
Church bells ring	2	Abs	Remaining Time: +57:42	
Reinforcements Arrive	3	Rel	Remaining Time: 2 rounds	on

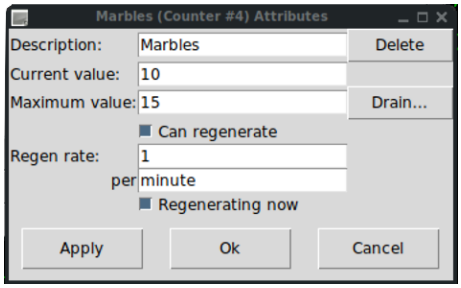
Event Description	ID	Value	Attributes	State
Marbles	4	5	(15) regen 1/minute	on
Church bells ring	2	Abs	Remaining Time: +57:42	
Reinforcements Arrive	3	Rel	Remaining Time: 2 rounds	on



For reasons that are not necessarily relevant today, if the counter's value drops to -10 or below, the color of the counter changes to red on a black background.[4]



As with timers, clicking on the right side of a counter will bring up an editor dialog which will allow you to change any of the parameters including the number of marbles currently in the bowl (say, if the PCs grab a handful on their way though the room).



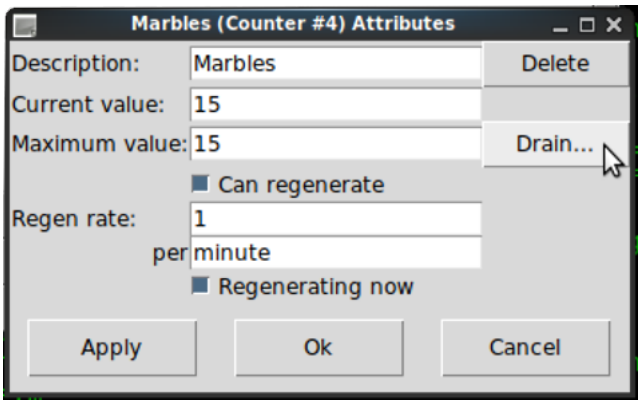
If you wish to temporarily suspend the regeneration effect, just un-check the "Regenerating now" box or click the "on" or "off" in the event list (as previously demonstrated for timers).

Drains

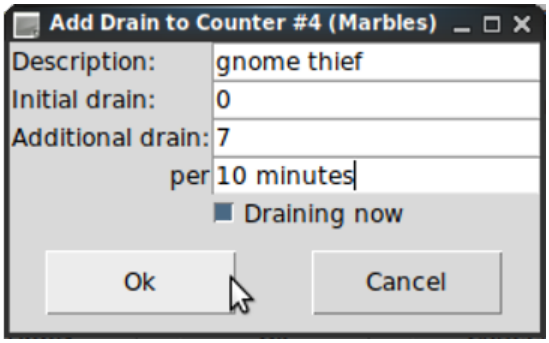
Now the plot thickens. You see, every ten minutes, a gnome runs into the room, steals seven marbles, and runs away. Whatever sort of thing a counter is keeping track of, there are many situations where, just as the counter itself may regenerate its value over an interval of time, some other effect may be going which drains away its value over time.

This is accomplished by creating another event called a *drain*. Unlike counters and timers, drains are not placed directly on the event list but are rather attached to a counter. To create a drain on our marble counter, we'll click on the attributes section of the counter in the events list to bring up the editor dialog for that counter.

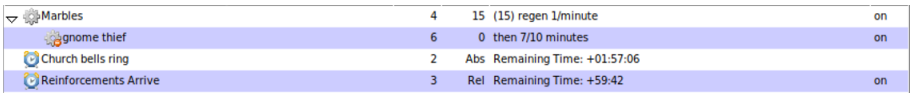
[4] This is from the days when we used these to keep track of hit points.



This dialog contains a Drain... button to create a drain for this counter. Click on that. Now we are given a dialog to enter the drain's details.

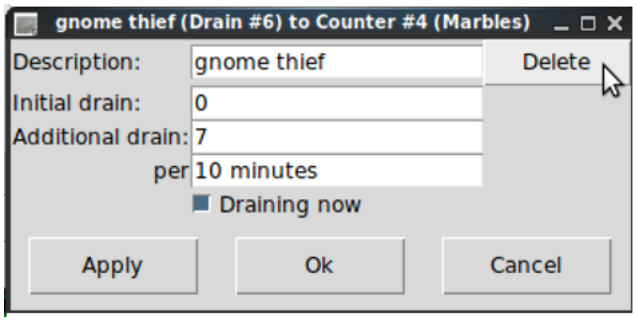


The "Initial drain" value is a number to immediately deduct from the counter's value. This is a one-time reduction. In this case we don't need one so we enter 0. In "Additional drain" we indicate the amount by which the counter will be reduced followed by the time frequency for that drain to occur. For this example, we ask for a drain of 7 marbles every 10 minutes.



Note how the drain is listed as being attached to its counter by being indented and immediately under the counter. If you click on the triangle next to the counter, the list of drains attached to it is collapsed so they are not displayed. Clicking the triangle again expands the list back out so they are visible.

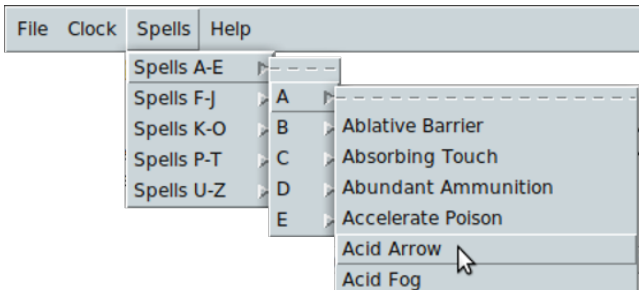
As with the other events, clicking in the attributes area of a drain brings up a dialog to edit the parameters of the drain. You can check or un-check "Draining now" to temporarily suspend the drain's effect (or click to toggle the "on" and "off" state in the event list). Or you may remove the drain entirely by clicking Delete on this dialog.



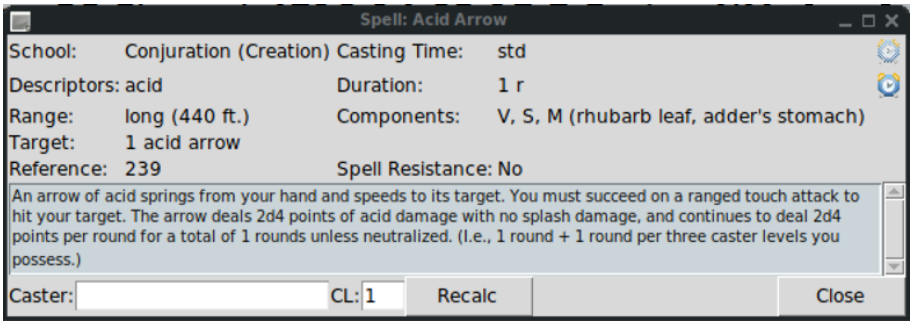
Spells

The GMA console provides quick access to the built-in database of spell descriptions, to make it easy for the GM to look up their effects. It also ties spell-casting times and durations into the event timer system.

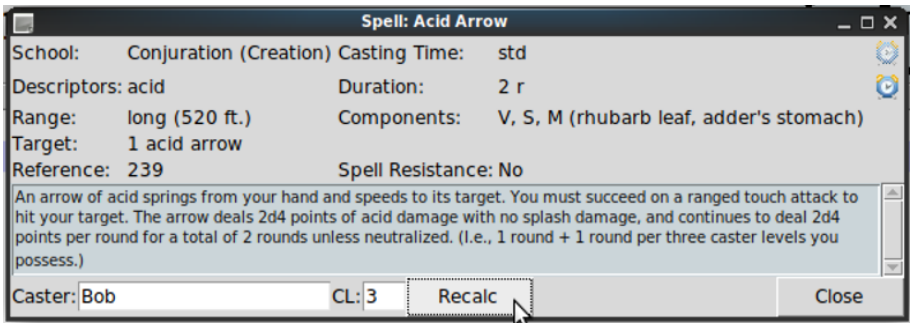
Let's say Bob the Wizard wants to cast *acid arrow*. The GM may look it up by selecting Spells › Spells A–E › A › Acid Arrow from the menu.



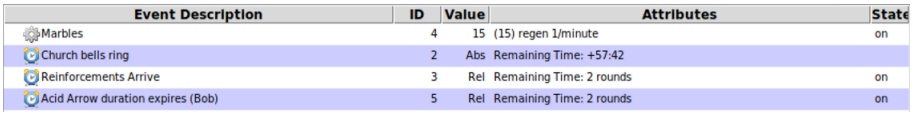
This brings up a basic description of the spell.



Since Bob is third level, we update the caster name and caster level (CL) then click Recalc to have the variable parameters in the spell description adjust for that level of caster.



Note the timer icons next to the spell casting time and duration. Because the casting time is one standard action, the timer button is disabled, since that's not something that will take time to accomplish. The duration, however, is 2 rounds. If we click the timer button we will get an automatically-generated timer event which will notify us in 2 rounds when the spell wears off.



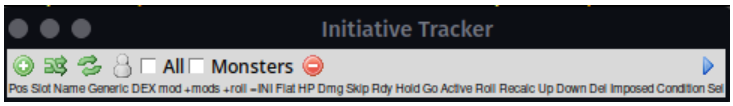
Running Combat

"Roll for initiative!"

With those words, the GM indicates that things just got exciting for the adventuring party. They may be engaged with a pack of enemies, fighting for their lives in an all-out *mêlée*. Or, perhaps they just triggered a trap and have seconds to escape from a locked room while it is filling with water. And spiders.

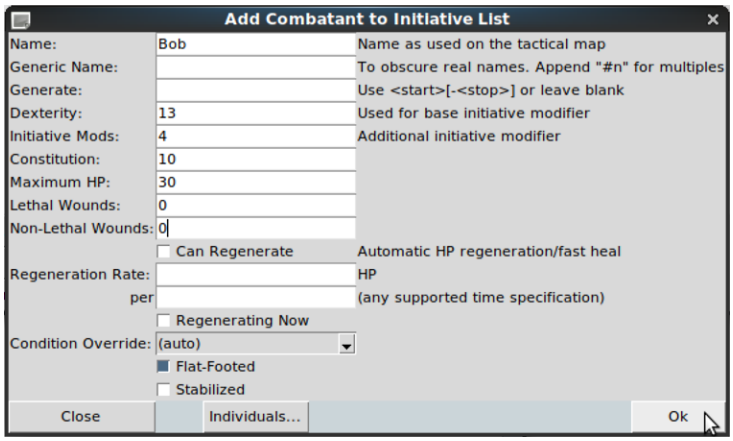
Whatever the situation, when the game moves from free-form role playing to combat, time effectively stops while the GM figures out who can act first, and who gets to act next, and so on, adjudicating the events of every second in precisely-timed detail. It can be a great deal of tedious work, and is one of the main reasons GMA was created.

To set up the initiative list, select Clock » Manage Initiative... from the menu or click the Manage Initiative icon ( ) on the toolbar. This brings up two new windows. One is the large clock display, which we can set aside for the moment. The other is the initiative tracker.



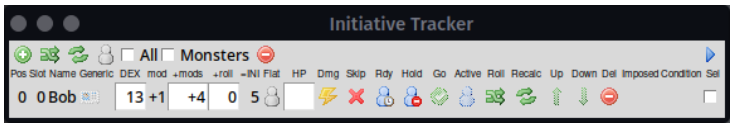
This is where we will keep track of the order in which each participant gets to act. First, we need to add everyone to the tracker. Click the "Add combatant" icon ( ) in the upper-left corner of the Initiative Tracker. In

the dialog that appears, fill in the details for a combatant, such as Bob the Wizard.

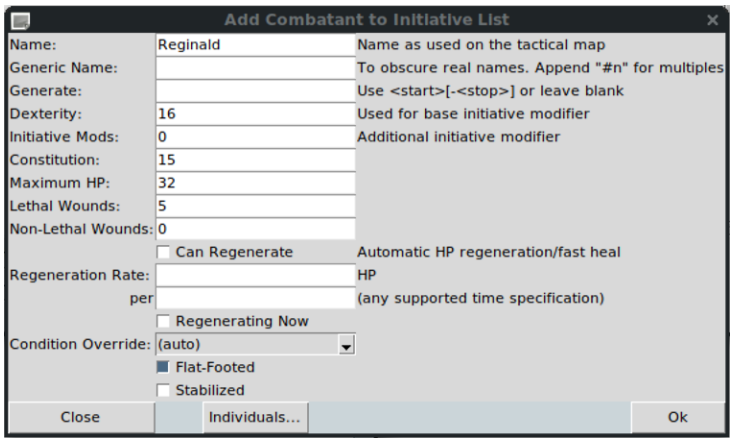


Since Bob has a Dexterity score of 13 and also has the Improved Initiative feat which gives him a +4 bonus on initiative rolls, we enter 13 and 4 into the "Dexterity" and "Initiative Mods" fields. (The initiative modifier due to dexterity is automatically included for you. You just need to indicate how many additional points should be added to the initiative roll for the character.)

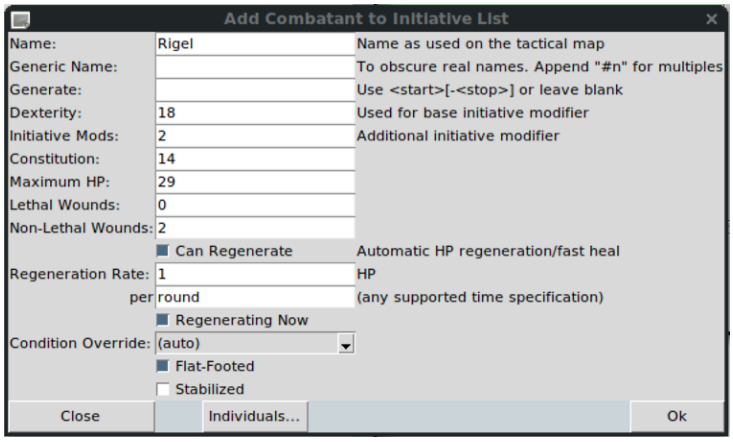
We also enter his Constitution score of 10, maximum hit point total of 30, and because he is at full health, we indicate that he has suffered 0 lethal and 0 non-lethal wounds against the 30 hit point total. When we click Ok, Bob will be added to the initiative tracker:



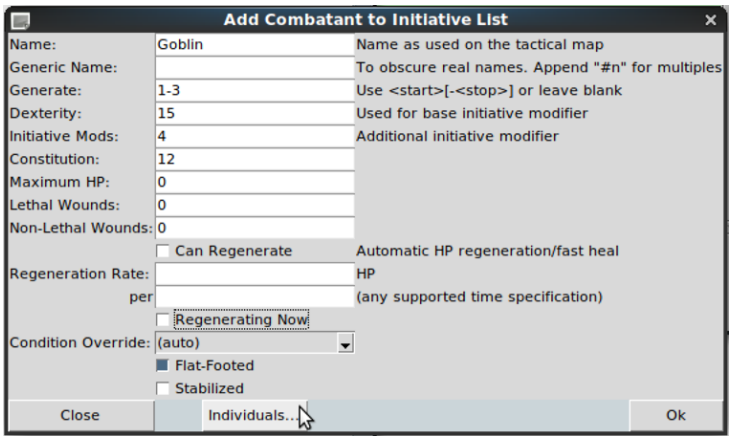
The dialog where we entered Bob's stats remains on the screen so we can use it to enter the next combatant. In this case, we'll add Reginald the Paladin, who has Dexterity 16, no additional initiative modifiers, Constitution 15, and 32 hit points. However, Reg has already suffered 5 points of damage so we note that under "Lethal Wounds", then click Ok to add him to the tracker.



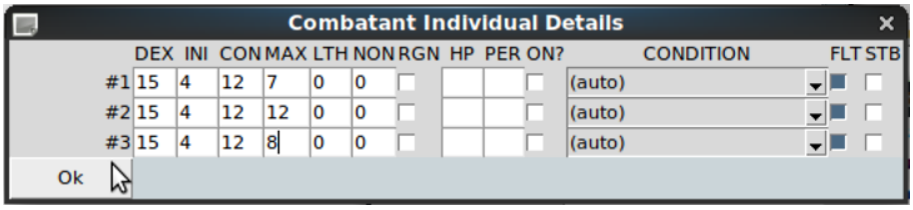
We repeat the process for the party's rogue, Rigel. She has an additional +2 bonus to initiative on top of what her 18 Dexterity gives her. She's also taken 2 points of non-lethal damage. Additionally, she has a magic item that heals her at a rate of 1 hit point per round, so we note that by checking "Can Regenerate," "Regenerating Now," and note the regeneration rate.



The party is up against three goblins. We can add all three at once to the initiative list since they share common attributes for the most part. As we fill out the dialog, we enter "`1-3`" in the "Generate" field.

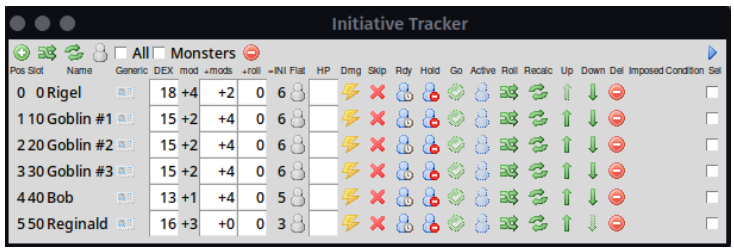


This will generate 3 goblins which will be named “Goblin #1,” “Goblin #2,” and “Goblin #3.” They have common stats which are entered as shown here. If we wanted to give them all the same hit points, we could simply enter that number here and be done with it. However, we wish to roll individual hit points for each one, so we enter 0 in the “Maximum HP” field (since it needs a valid input value regardless) and then click Individuals... to bring up an editor dialog that will allow us to adjust the stats for each of the goblins:



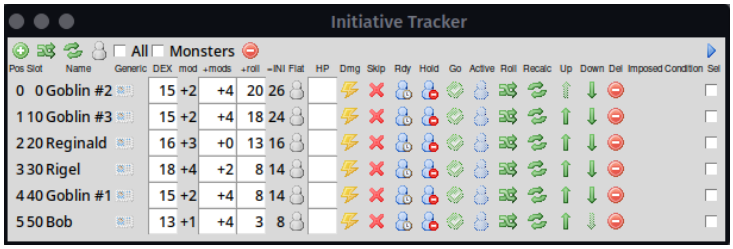
Here we enter 7, 10, and 8 as the individual hit point totals for the three goblins. We can also give each of them their own Dexterity and Constitution scores, initiative modifiers, lethal and non-lethal wounds, regeneration, and other condition status just as with the single combatant dialog. When we’re finished adjusting the individual stats, we click Ok to dismiss the individual editor dialog.

Once back at the single combatant dialog, we click Ok to add the goblins to the initiative tracker. Since we don’t need to add more combatants, we can close the dialog by clicking Close.



Now we see the six combatants on the tracker. We haven't actually rolled initiative yet so they are simply in order of their initiative modifiers. Looking at the first few columns, we see the position of each, numbered 0–5, showing the order in which each person can act. Their initiative *slots* are also listed. A combat round is divided into 60 slots (corresponding to the "count" interval we discussed earlier). GMA evenly distributed the combatants through the span of the round, so they act in slots 0, 10, 20, 30, 40, and 50 in this case.

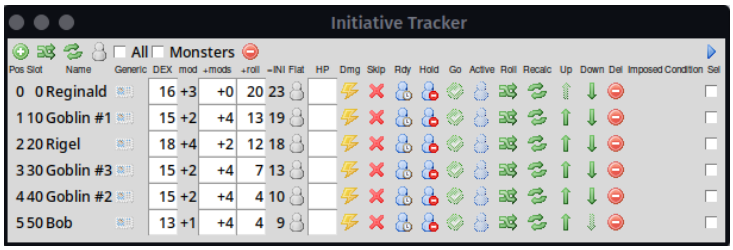
If we wish to automatically roll initiative for everyone, just click the roll button ( ) which will re-sequence the initiative order according to those die rolls:



You can also see each person's dexterity, other modifiers, and the initiative die roll made for each. These are data entry fields, so they may be altered as needed over the course of the combat. If any of the values are changed, click the refresh button ( ) to have the initiative tracker take the changes into account and re-sequence the list accordingly.

N.B. Clicking on the refresh button ( ) at the top of the tracker resets *everyone's* position in the list based on their stats, undoing any manual rearrangement you may have made. Clicking the recalc button ( ) on a particular line of the tracker resets *only* that person's position.

For example, if the players want to roll their own initiative, have each of them roll a d20 and report the result. If Bob, Reg, and Rigel roll 4, 20, and 12 respectively, and we say the goblins rolled 13, 4, and 17, we update the tracker with that information and click the refresh button.

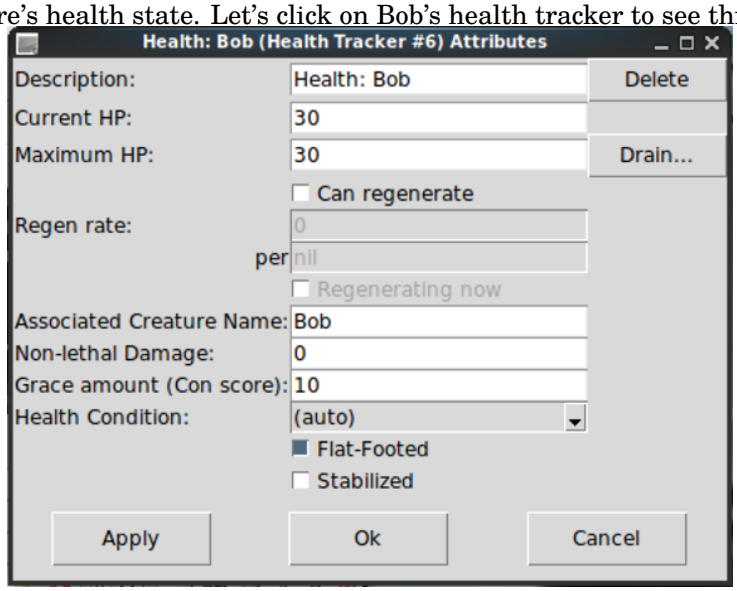


Health Trackers

Looking back at the event list, we notice that GMA has added more trackers as we added combatants to the initiative tracker:

Event Description	ID	Value	Attributes	State
Marbles	1	15	(15) regen 1/minute	on
gnome thief	2	0	then 7/10 minutes	on
Health: Bob	6	30	/ 30 [FLAT]	off
Health: Reginald	7	27	/ 32 [FLAT]	off
Health: Rigel	8	29 (2)	/ 29 [FLAT] regen 1/round	on
Health: Goblin #1	9	7	/ 7 [FLAT]	off
Health: Goblin #2	10	10	/ 10 [FLAT]	off
Health: Goblin #3	11	8	/ 8 [FLAT]	off

These new items are *health trackers* which are like an enhanced form of a counter. The value they keep track of is the hit point total for a particular creature. Just like regular counters, if we click on the "Attributes" part of a health tracker, we get a dialog which allows us to edit the details of that creature's health state. Let's click on Bob's health tracker to see this:



If Bob's hit point total changed, or he became stabilized or flat-footed, we can make those changes here. We can also enter a regeneration rate for his hit points, just like with regular counters. Also of interest is the fact that we can also create a drain on these hit points just as we do with counters.

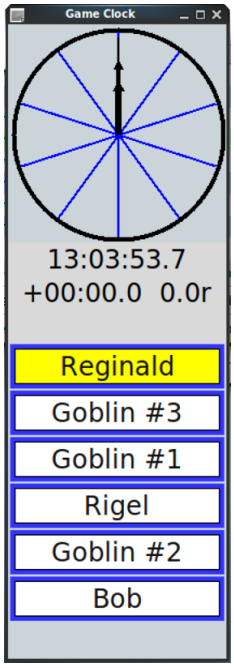
By entering the name of a creature on the initiative tracker in the "Associated creature name" field we maintain a linkage between that creature and this health tracker.

We can also manually create a health tracker which may or may not be associated with the initiative tracker by clicking on the health tracker button (❤) on the toolbar or choosing Clock » Add Health Tracker... from the menu.

Like the simpler counters, health trackers also change background colors as the hit point total depletes toward death. However, they also know to report creature death based on the rule that a creature may drop to a negative hit point value equal (in absolute value) to the creature's Constitution score. While less than 0 HP, the creature "bleeds" 1 HP per round. When a

creature drops below 0 HP during combat, GMA automatically places a drain on that health tracker to facilitate this bleed effect without overt action on the GM's part.

Managing Initiative and Events in Combat



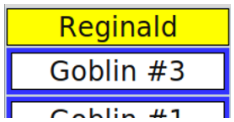
Select Clock > Reset Elapsed Time or click the Reset button of the clock so that our elapsed time counter is tracking time since the start of combat. Then click the Combat button below it or select Combat > Toggle Combat Mode to put GMA into combat mode. The clock face changes from displaying the current time of day to a special stopwatch-style display.

In this mode, the longest hand sweeps around the clock once each round. The next hand sweeps once per minute, and the third makes the cycle once per 10 minutes. Looking at the large clock window we set aside earlier, we see that under the clock face, GMA has listed the combatants in order of when they can take their turn.

Everyone begins combat flat-footed, which is shown in their health tracker with a "[flat]" notation (or with the "Flat-footed" box checked in the editor dialog). On the initiative clock display this is indicated by a heavy blue border around the combatant's name.

To toggle the flat-footed status for an individual, click the flat-footed button ( ) on their line of the tracker. To set everyone to flat-footed at once, click the flat-footed button on the initiative tracker's toolbar or select Combat > Set All Flat-Footed.

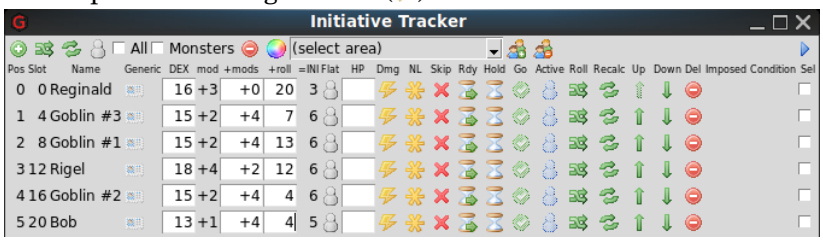
To start combat, we need to advance to the first active initiative slot. Since in our example, Reg is in slot 0, and our clock is at 0 now, we're already there. We can see that his slot on the initiative list under the large clock is highlighted with a yellow background, indicating that it's his turn. If that wasn't the case (i.e., the first combatant was in a slot after #0), click the "next slot" button ( ) in the upper-right corner of the initiative tracker to advance the clock to the next slot where someone is able to take an action. The >>ct button on the main GMA window and the Combat > Advance to Next Slot menu item does the same thing.



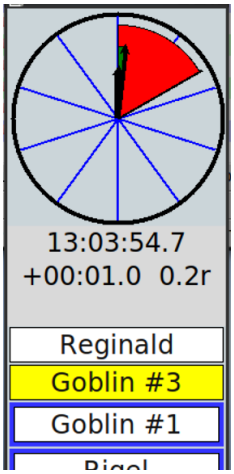
Since Reg can now act, we click his flat-footed button ( ) to indicate that he is no longer flat-footed. The blue border around his name disappears.

Reg decides to attack Goblin #1, successfully hitting it for 3 points of damage. Instead of opening up the goblin's health tracker to change the hit point to-

tal, the initiative tracker provides us with a single place to record wounds against any combatants. Just enter the wound in the "HP" column for Goblin #1 and press the "Dmg" button (⚡):



After clicking the "Dmg" button, the goblin's health tracker shows the loss of hit points.



If you want to record non-lethal damage, follow the same procedure but click the "NL" button (✳) instead.

In either case, instead of simply typing an integer as the number of lethal or non-lethal hit points to deduct, you can write a more complex expression enclosed in square brackets, such as "`[15+2+5]`" or even "`[2d6+12]`" if you'd like GMA to roll damage dice for you and deduct the result from the creature's hit point total all in one step. More formally, if you enclose the damage value in square brackets, it will be evaluated by GMA's internal die-roller expression engine to resolve random die rolls it contains. This also means it will help you add up several numbers or do other basic math functions for you even if you don't include any actual dice in the expression.

If you enclose the input in double square brackets (e.g., "`[[1d6+4]]`"), then the expression is sent to the GMA server (assuming you're connected to one) instead of evaluating it locally. If you want to do this but keep the results private ("behind the GM screen," as it were) then place a question mark just after the brackets, like so: "`[[?1d6+4]]`".

Now it's the next person's turn. We click the "next slot" button (▶) to move the clock forward to the next slot, which is Goblin #3's turn. Since they can now act, we click the flat-footed button for Goblin #3. The clock now looks as shown on the right.

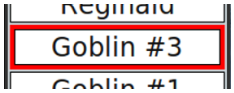
Goblin #3 scores a critical hit for 11 points against Rigel, so we enter 11 in Rigel's "HP" box and click the "Dmg" button, then click the "next slot" button to advance the clock to Goblin #1's turn. Finally, we click the flat-footed button for Goblin #1 so he can act. He tries to attack but misses. We click "next slot" again to advance to Rigel's turn and click to turn off her flat-footed status as well.

Rigel scores a hit for 8 points against Goblin #3 in retaliation for his earlier attack on her. We enter 8 in Goblin #3's HP box and click the "Dmg" button next to it to remove that many hit points. Unfortunately for the

goblin, that's all the hit points he had, so his hit point tracker turns red to indicate that he's now disabled or dying:

Event Description	ID	Value	Attributes	State
Marbles	1	15	(15) regen 1/minute	on
gnome thief	2	0	then 7/10 minutes	on
Health: Bob	6	30	/ 30 [FLAT]	off
Health: Reginald	7	27	/ 32	off
Health: Rigel	8	18 (2)	/ 29 [FLAT] regen 1/round	on
Health: Goblin #1	9	4	/ 7	off
Health: Goblin #2	10	10	/ 10 [FLAT]	off
Health: Goblin #3	11	0	/ 8	off

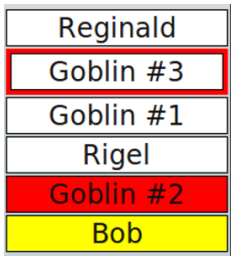
On the initiative list, a heavy red border is drawn around the goblin's name to indicate this status as well:



Moving on, we click the "next slot" to advance the clock to Goblin #2's turn and remove his flat-footed status.

Goblin #2 is a little freaked out by his friend just getting dropped and realizes that Bob looks like a magic-user so he's going to ready an action to attack Bob as soon as he starts casting a spell.

A *readied action* is different than simply holding one's action in that it has a specific trigger he's waiting for before taking his action. Because of that, when the trigger condition happens, he can act *first*, interrupting the other person's turn to go before they do.



To note that the Goblin #2 is readying an action, we click on the "Rdy" button ( ) on his line of the initiative tracker, and then click the "next slot" button to advance to Bob's turn, also removing his flat-footed state.

On the initiative list displayed under the clock, we see that the goblin's name is marked in red to indicate the readied action.

Bob, for his part, eyes Goblin #2 suspiciously, seeing him poised to attack him, so he decides to hold his action while the two of them just stand there looking at each other. Since this is not a readied action, merely a regular *held action*, we click on the "Hold" button for Bob ( ).

That's the end of the round. We click the "next slot" button to advance the clock back around to the start of Round 2. We also see that Bob's name is also marked to show that he's holding his action, as shown in Figure 3.4.

Reg attacks Goblin #1, scoring 6 points of damage against it. We enter 6 in the "HP" box and click the "Dmg" button. This puts the goblin at negative hit points, so it is down and bleeding out. Looking at the health tracker for the goblin, we see that GMA has automatically attached a drain to remove 1 HP/round due to this:

Health: Goblin #1	9	-2	/ 7
DYING (bleed-out)	14	0	then 1/round

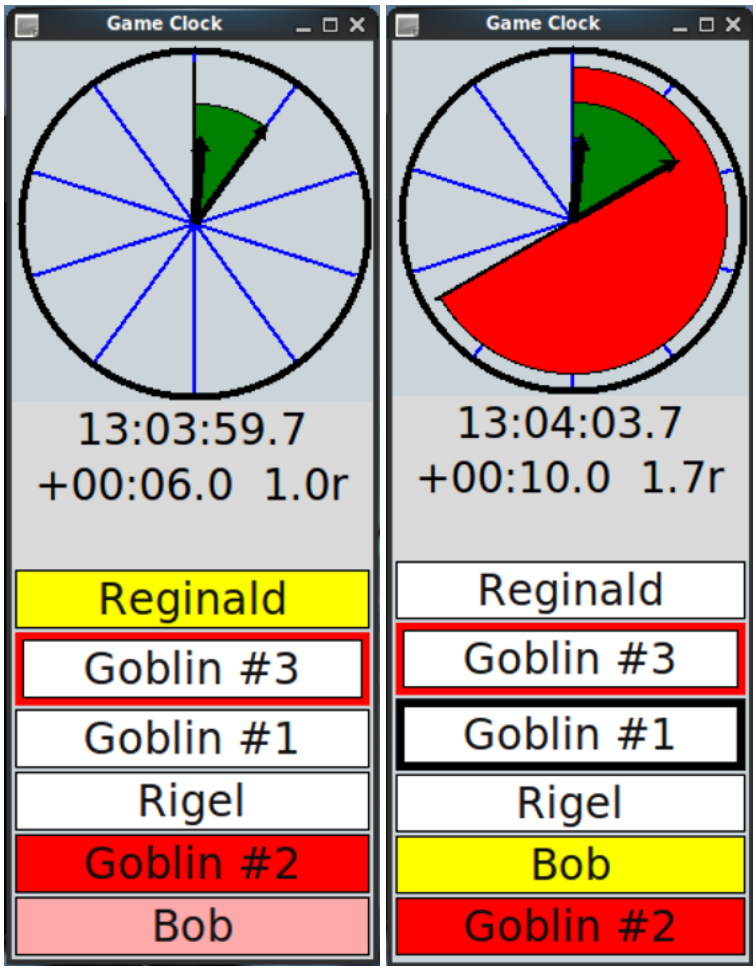
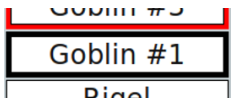


Figure 3.4: Initiative Clock Display at Start of Round 2 (left) and After Bob's Held Action (right).



In the initiative list, the dying goblin's name is shown with a black border to show that it is effectively out of the combat now.

Since Goblin #3 is disabled and Goblin #1 is dying, we'll click the "next slot" button a couple of times to skip past them and move on to Rigel's turn. She takes a step back from the battle and drinks a healing potion, restoring 6 HP. We note that by typing "-6" in her HP box and clicking the "Dmg" button.

Now Bob wants to move, taking his held action now. We click the "Go" button ( ) on his row of the initiative tracker to indicate that he will go now. This becomes his new position in the initiative list, as we see in Figure 3.4. Using his mace, he delivers a *coup de grâce* on the dying goblin, dealing 12 points of damage, enough to kill it outright. Since it's dead, its health tracker turns to red on a black background. GMA doesn't automatically assume it can turn off the bleed-out drain, however, but you can manually turn it off.

Advancing the clock to the surviving goblin's turn, he realizes that it's pointless to continue to hold his action so he lets go of that. By clicking the "Active" button ( ) it returns him back to normal action status, cancelling either a held or readied action, without changing his position in the list. For his action, the goblin makes a desperate attempt to intimidate the party, but fails.

We click the "next slot" button again to advance to the start of Round 3. Since we have a couple of dead combatants, we can clean up the list by telling GMA to skip over them by pressing the "Skip" button ( ) for the two deceased goblins. This will cause them to be removed from the displayed list, and the "next slot" button will skip over them, but all other processing such as drains and regeneration will still take place. Figure 3.5 shows the clock at the beginning of the round before and after we skip the goblins.

The remaining buttons on the initiative tracker perform the following functions:

Roll Roll initiative for this combatant, updating their position in the initiative list accordingly. This is useful when adding new combatants to the fight after the first several have been rolled.

Recalc Recalculate the initiative position for this combatant. If you make any changes to the values on this line, this button will cause them to take effect without changing any other combatants.

Up Move this combatant up one position in the list (to act earlier).

Down Move this combatant down one position in the list (to act later).

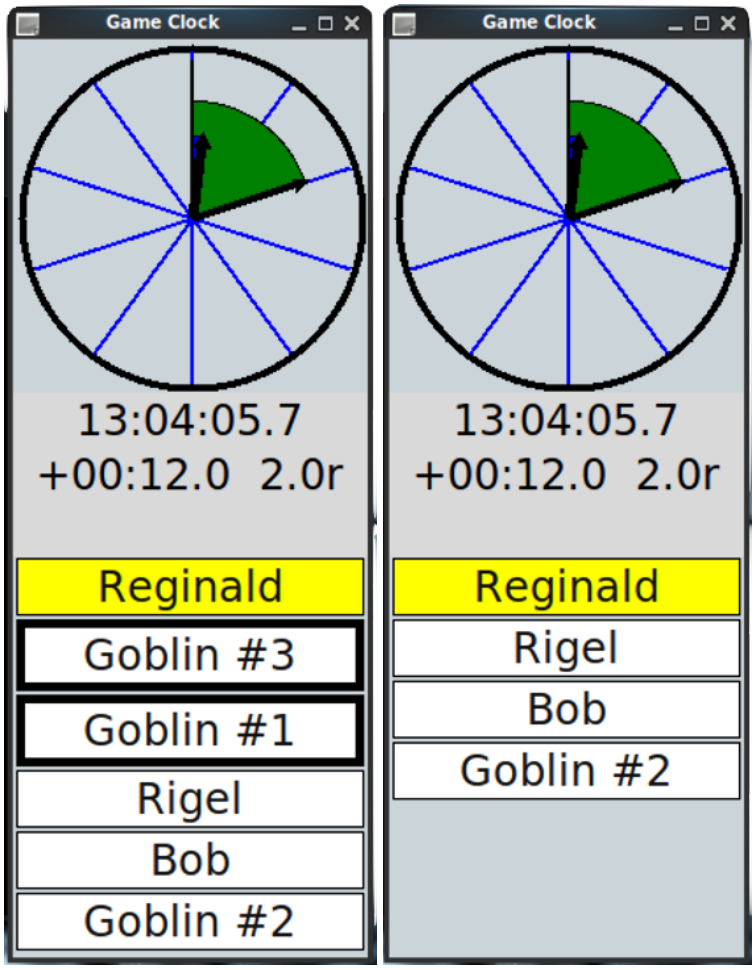


Figure 3.5: Initiative Clock Display at Round 3, Before and After Skipping.

Ending Combat

When combat is over, click on the "Del" button (⛔) for each person you wish to remove entirely from the initiative list. You should leave in anyone (such as the PC party) who will continue on to engage in further adventures, so you won't need to enter their stats again (and they can continue to heal, regenerate, etc.). When you delete them from the initiative list, their health tracker is automatically deleted from the event list in the main window as well.

You can remove many combatants at once by checking the boxes at the far right of their rows in the initiative tracker, then pressing the delete button at the top of the initiative tracker window. For convenience, at the top of the window are two checkboxes. The one labelled "All" toggles on or off all of the combatant's checkboxes, while the one labelled "Monsters" does the same but only for the monsters.

Note that when you click the "Del" button, it removes the combatant(s) from the initiative list *and* their health tracker from the main window. If you hold the shift (⇧) key down when you click the "Del" button (⛔), they will only be removed from the initiative list. This leaves their health tracker intact so they can show up later in a different encounter with accurate health statistics at that time. If, for example, they are regenerating, their hit points will automatically increase over time.

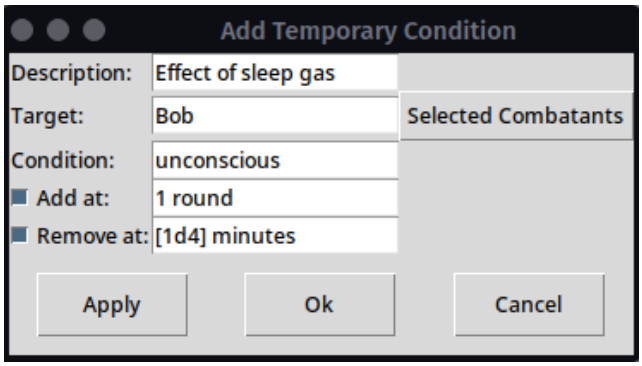
If you are using a `.gmo` file to pre-define the combat encounters[5] and have a monster who is escaping combat but is likely to re-enter combat again at another place and time, see page 155. As long as you add them back to the initiative tracker with the same name, GMA will link that creature to the existing health tracker of the same name.

Take GMA out of combat mode by clicking the Time button to the right of the clock face on the main window or selecting Combat › Toggle Combat Mode. This will return the clock display to show the time of day again.

Condition Timers

One other thing you can do with the timekeeping mechanism in GMA is to track the timing of *conditions* being placed on characters. Clicking on the "Add Condition" button (⚠) in the main window's toolbar or selecting Clock › Add Temporary Condition... from the menu brings up a dialog for you to enter a condition to be applied.

[5] See Chapter 7.



Bob has unluckily walked into a room full of sleep gas. In one round he will fall unconscious and remain unconscious for 1–4 minutes (determined randomly by rolling 1d4). We fill in a description of what this condition timer is for ("Effect of sleep gas"), and the name of the target ("Bob"). The target name should exactly match the name of someone on the initiative tracker. (But it isn't a requirement—if it does match someone on the list, GMA will be able to update their condition as shown on the map as we will see in a later chapter; however, if no such person is in the initiative list, then this is just a note to yourself about the condition change.)

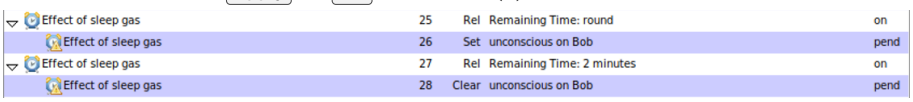
If you want to simultaneously apply a condition to a number of creatures, you can type a list of names in the dialog box instead of a single target, with commas between each name. As a handy short-cut, you can instead check their boxes (at the far right of their rows in the initiative tracker window) and then click the Selected Combatants button instead of typing a list of names manually.

Then we list the condition that will take effect ("unconscious"). If we check the "Add at:" box and fill in an onset time ("1 round"), then a timer will be added to our event list which will expire at that time, indicating that the condition should be added to the target (Bob succumbs to the gas and falls unconscious). If the box is unchecked, it is assumed that the condition is applied immediately.

If we check the "Remove at:" box and supply a time interval ("[1d4] minutes"), a(nother) timer will be created which expires at the end of that time, indicating when the condition should be removed (Bob regains consciousness).

Note the way we specified the latter time interval. Placing a random die-roll expression in square brackets tells GMA to roll those dice for us and use the result in that place of the time value.

When we click Apply or Ok the timer(s) are added to our event list:

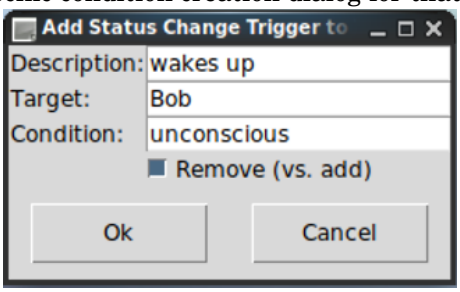


These act like the timers we discussed at the beginning of this chapter, except they also show that a condition of a character is associated with the

timer's duration.

On the far right column of the condition is the status "pend" which means the condition change is pending (waiting for the timer to expire before taking effect). When the condition is applied this will change to "DONE." Clicking on this status indicator will flip it to "BLK" which means to temporarily block the status change from taking place.

You can attach one or more condition changes to any timer, in fact. If you click on a timer to get its editor dialog, you can then click on the Condition... button to get a specific condition creation dialog for that timer:



We don't enter timing information here since the timer already exists; we only need to enter information about the condition. Check the "Remove" box if this timer is to remove the condition from the target. Uncheck it if the condition is to be applied when the timer expires instead.

Endgame

When finished with GMA, select File > Quit from the menu to exit. The current time of day and the event list (timers, counters, and so forth) are saved in the game database so the next time you start GMA it will be in the same state as when you last used it.

If you were in the middle of combat when quitting, you should save the initiative list to a file before exiting GMA. Do this by selecting Combat > Save Initiative List As... from the main menu. Then, when resuming play, open the initiative tracker and reload its contents by selecting Combat > Load Initiative List....

Calendar Type

The default calendar is that used by the Golarion campaign setting for Pathfinder. Other calendars are supported. To change this, edit the calendar setting in your campaign world preferences. Currently-supported choices include:

`donuttus` The calendar for the toroidal planet Donuttus invented by the author and his friends in high school. The calendar consists of 12 months (inspired partially by the Donuttus mythology, the Zork games from

Infocom, and probably other things) called Lith, Estuary, Elwinder, Umbar, Ektar, Balthmaar, Frobuary, Trovuary, Solmark, Wedmark, Blotmath, and Aberlith. Each month has 28 days, arranged into 7-day weeks. The weekday names are Sunday, 1stday, 2ndday, 3rdday, 4thday, 5thday, and 6thday. The four seasons, spring, summer, fall, and winter are three months each in length and begin with the first day of Lith as the first day of spring.

The moon, which is really a strange metallic object in the hole of the torus, changes color throughout the year: red, yellow, white, green, cyan, blue, and black (dark). Each color lasts for 1 week.

`golarion` The calendar used by the official game world of the Pathfinder role playing game. It is based mostly on the Gregorian calendar, although months and weekdays are renamed.

`gregorian` This is the familiar calendar used as the current world-wide standard (for the most part) on Earth. We assume it needs no further explanation.

`tien` A variation of the Golarion calendar with the year shifted in time to match the calendar of the Golarion continent of Tian Xia. It is exactly 2500 years ahead of the standard Golarion reckoning used elsewhere in the world.

CHAPTER 4

Introducing the Mapper

> What is a fantasy map but a space beyond which There Be Dragons?
>
> —Sir Terry Pratchett

Since the dawn of fantasy role-playing games, nay, since even before the first people played Dungeons & Dragons—in the days of wargames with miniature figures—players have used tactical maps to visualize the positions of their combat units, the surrounding terrain, obstacles, and so forth.

In the earlier days of the author's own D&D campaigns, we printed maps on large sheets of paper using a Perl script called `mkmap` which showed environmental features (trees, dungeon corridors, walls, doors, furniture) on a scale suitable for placing standard miniature figures onto. As the party explored, the GM would take another cut-out section of the paper map and tape it onto what was already on the table, so the players could see more and more of the dungeon as they explored.[1]

Our needs have evolved along with the increasing availability and power of computing resources over the years since that time. Now we project a digital battle map on a wall or large monitor screen during our games, using the GMA `mapper` program to display the tactical map (or "battle grid" as some call it). Player characters, NPCs, and monsters are shown on the map

[1] This program is still included in the GMA suite in case it is of interest. See the documentation on p. 500 for details.

as movable tokens instead of picking up and moving physical miniatures on the older paper tabletop maps.

This chapter will introduce the `mapper` program and demonstrate its capabilities out-of-the-box (as we did in Chapter 3 for the GM console). In later chapters we will see how a little advanced preparation can greatly improve the experience of using the mapper during your games.

Appendix I on page 725 provides an introduction to using the mapper program from the point of view of players who wish to use the map themselves to move their character tokens around in interactive networked mode (which we'll set up later, in Chapter 9). Since it gives a good first look at the mapper and the various things you can do with creature tokens, go read that first and then come back here to learn about all the other things you can do as the GM to use the map when running your sessions. Don't worry about the networked multi-user bits until we get to Chapters 9 and 11.

Creating Basic Maps

Now that you've gone through the tutorial in Appendix I, you have seen how to start the mapper tool and gone through the basics of placing a character token on the map, zooming in and out, and so forth.

As the GM or cartographer in your group, you'll need to also use the mapper to create the maps that will be seen during your game. We will step through each of the drawing tools in this chapter, showing how to use them to draw simple floorplan maps. We will then build on this level in Chapter 5 to add nicer graphic images to the map.

Setup for Map Creation

Before you proceed to create or edit map files to be used in your games, you need to make sure that the Edit ⟩ Force Drawn Elements to Top menu item is *not* checked (Figure 4.1).

This is normally checked during an active game, so that any objects you draw manually onto the map to note the location of objects, spells, etc., are always placed on top of the other map elements. However, this needs to be off when creating the base map itself. Otherwise, everything is forced to the top, which defeats the entire purpose of this menu item.

Basic Drawing Tools

When you start the mapper, you can see the controls spread out across the toolbar at the top of its main window.



These are arranged into related groups of tools. We'll start with the group on the left, which provide a set of basic drawing tools for putting geometric shapes into the map. Clicking on one will select it, deselecting the previous tool that was in effect.

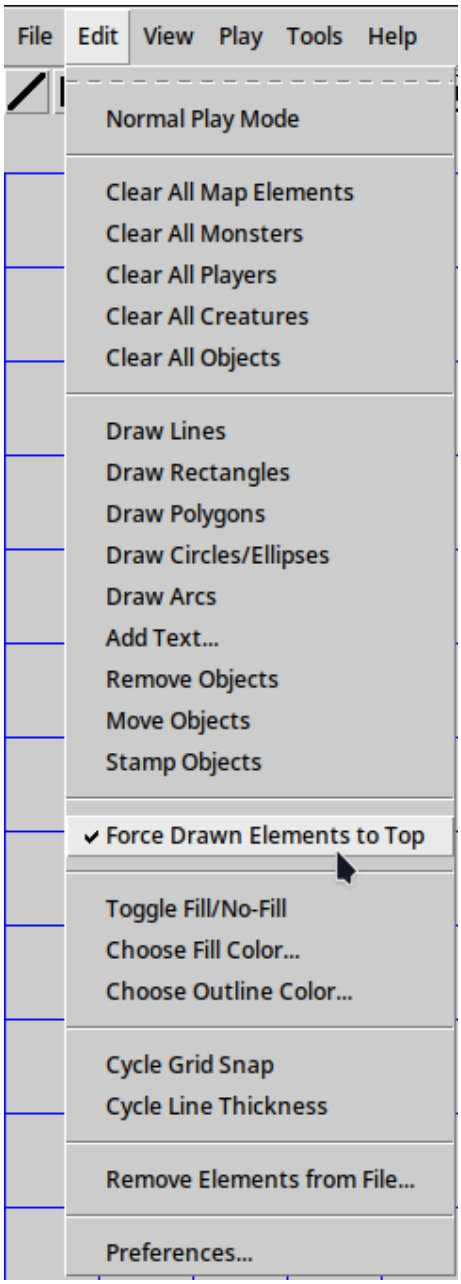


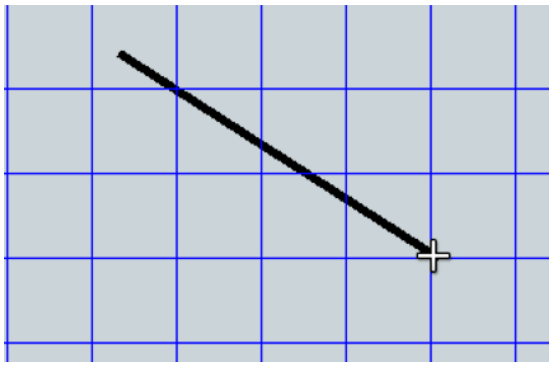
Figure 4.1: The Force Drawn Elements to Top Menu Item



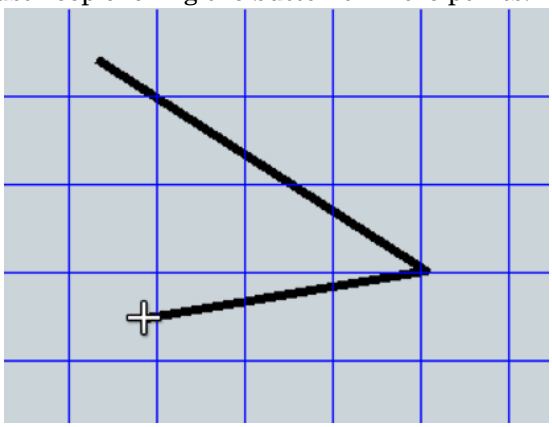
From left to right, these are the line, rectangle, polygon, circle, arc, and text drawing tools. The three blank buttons are used to select options and modes specific to the drawing tool currently in effect (so they will change based on which tool is active). Then we have the normal play mode tool (which you already saw in the tutorial in Appendix I), and the delete, object move, and stamp tools.



To draw straight lines, select the line tool, then click the left button on the point where you want the line to start, then drag the mouse to the next point (there is no need to hold down the button while dragging between points). Click the mouse again on the ending point for the line.



If you wish to keep drawing more lines between that point and another, and another, just keep clicking the button on more points.



After clicking on the last point, press the Esc key to complete the line object. If you click the button again, you will begin drawing a new line object.

With the line tool selected, two option buttons become available. The first controls arrowheads, with these possible settings (click repeatedly on the button to cycle through these choices):

— Draw line segments only, without any arrowheads.

← Draw an arrowhead on the first point only.

→ Draw an arrowhead on the last point only.

↔ Draw arrowhead on both ends of the line.

The second option button controls whether the line is drawn with solid or dashed lines. Clicking on this button cycles through these choices:

— Draw solid lines.

-- Draw lines with long dashes.

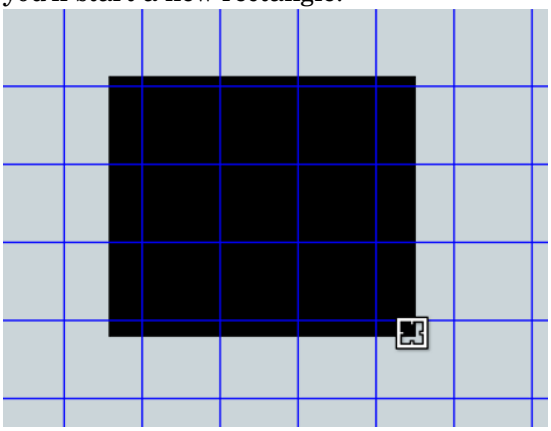
- - Draw lines with short dashes.

···· Draw lines with tiny dashes (dots).

-·- Draw lines with alternating long dash-dot pattern.

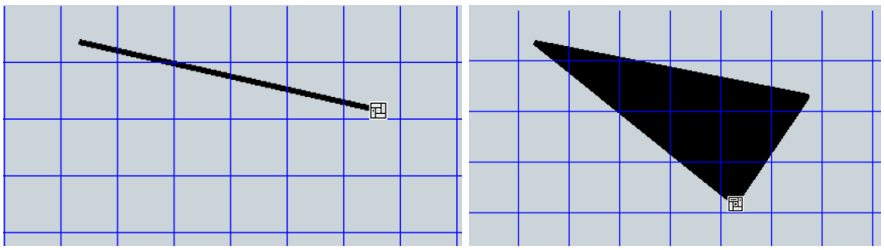
-··- Draw lines with alternating long dash-dot-dot pattern.

Drawing rectangles is accomplished by selecting the rectangle tool, then clicking on two diagonally opposing corners. When you click on the second corner, the rectangle object is complete. If you click on another point, you'll start a new rectangle.



The same dashed-line pattern options as described above for lines are also available if you want dashed lines for the perimeter of the rectangle.

Polygons work just like lines, as described above, but the first and last points are connected to create a closed polygonal area, which is filled in with a solid color. As with lines, click the button on each point around the polygon's perimeter and then press Esc to complete the object.

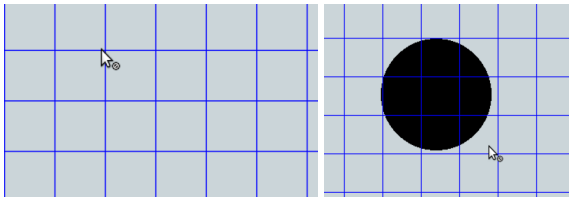


The options available for polygons include the same dashed-line option as with lines and rectangles, in addition to a join-style selector, which cycles between these options:

- Beveled corners at each vertex.
- Mitered corners at each vertex.
- Rounded corners at each vertex.

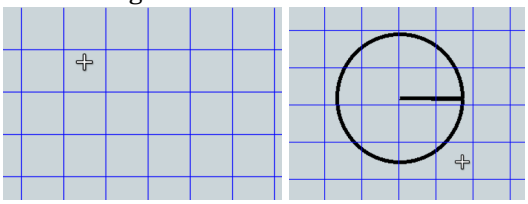
There is also an option selector to control the degree of spline curves to make for the polygon's sides. The default is which indicates that straight lines should be used. The other values range from which is the least degree of curvature to which is the greatest.

To draw circles and ellipses, select the circle tool and then proceed just like you were drawing a rectangle: click the left button on two diagonally opposing points. Rather than drawing the rectangle defined by those points, however, the mapper will draw the ellipse which is exactly inscribed inside that rectangle (so the rectangle's sides are tangent to the circle).

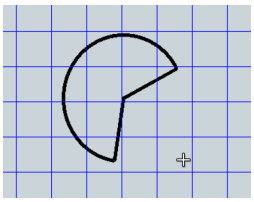


The same dashed-line option as with most other shapes is also available to allow you to make a circle or ellipse with dashed lines around its perimeter.

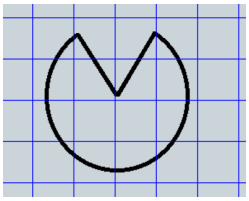
Arcs require a little more care due to the ways they can be positioned and proportioned. After selecting the arc tool, proceed to define a circle just as if using the circle tool.



Once you click to place the second corner, you will see the arc as a pie slice (by default; see below for other shape options).



Now you need to set the extent of the arc (how much of the circle's circumference should be included in the arc). You adjust this by moving the mouse vertically (without holding any buttons). At the same time, moving the mouse horizontally rotates the arc around its center. When the arc is rotated where you want it to be, and is of the desired extent, click the left button to commit the object.



You are also given the option to select between three different styles for the arc shape:

Pie-slice: the endpoints of the arc are connected to the center of the circle, forming a solid shape.

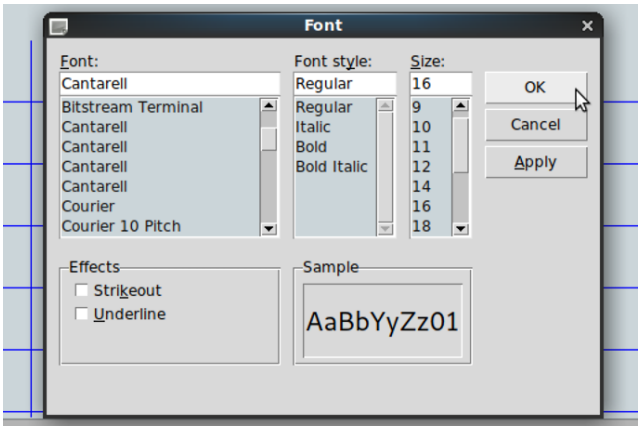
Chord: The endpoints of the arc are connected to each other with a straight line, forming a solid shape.

Arc: The endpoints are not connected.

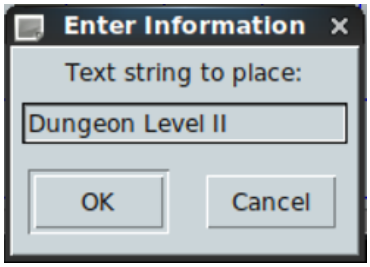
In addition, the same dashed-line option as with most other shapes is also available to allow you to make an arc with dashed lines around its perimeter.



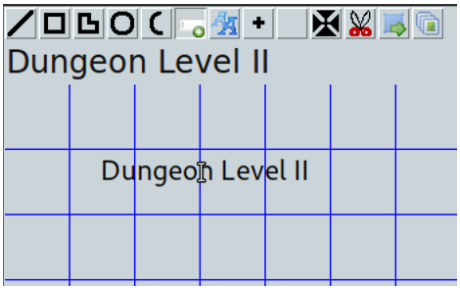
You can also place text onto the map using the text tool. With this tool selected, the font selection button () will bring up a dialog to allow you to specify the type font, style, and size to use for the text. (If you had just placed a text object, this dialog may be used to change it after the fact.)



The text tool is similar to the stamp tool in that it is designed to place a text object repeatedly on the map canvas. If there is no text string yet, or you wish to change it to a new one, right-click on the map canvas. You will be prompted for the text you wish to place on the map.



The entered text string is displayed in the information area just under the toolbar. Every time you left-click on the map, that text string will be placed at that location.



The option button for the text tool specifies how the text is located in relation to the point the mouse was clicked on:

- The text is centered on the point.
- The left edge of the text is centered on the point.
- The top-left corner of the text is at the point.

 The top edge of the text is centered on the point.

 The top-right corner of the text is at the point.

 The right edge of the text is centered on the point.

 The bottom-right corner of the text is at the point.

 The bottom edge of the text is centered on the point.

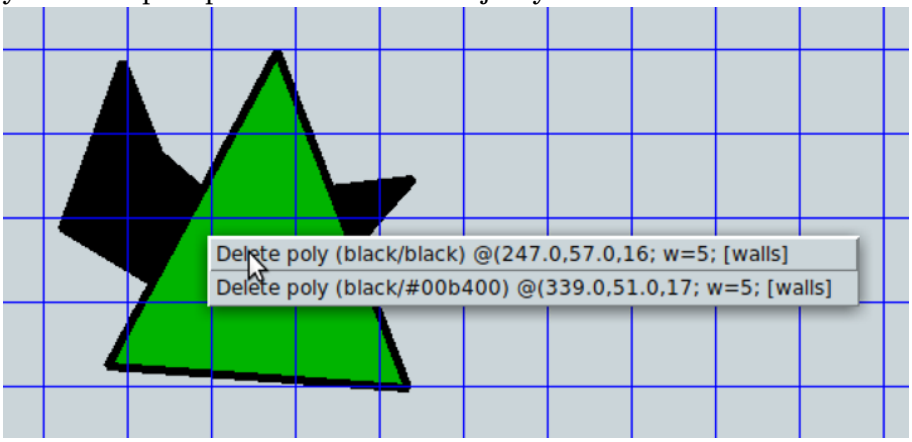
 The bottom-left corner of the text is at the point.



This is the default tool, and the one that should be selected most of the time during actual game play. With this selected, dragging a creature token with the mouse will move it to a new location. Right-clicking the mouse will bring up a menu of operations available in the context of whatever you clicked on. Dragging the mouse anywhere else will scroll the map canvas.

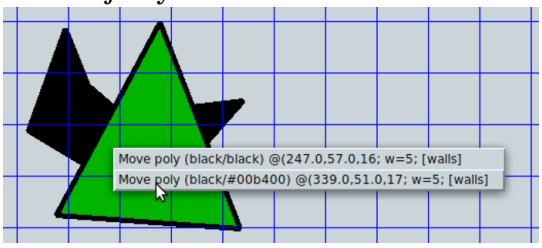


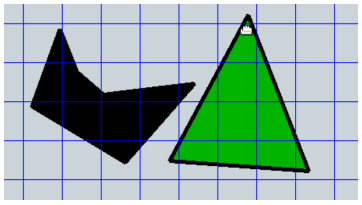
Selecting this tool allows you to delete any object on the map you click on. If more than one object is under the cursor where you click, you will be prompted to select which object you wish to delete.





Selecting the move object tool allows you to move any object on the map you click and drag with the mouse. If more than one object is under the cursor where you click, you will be prompted to select which object you wish to move.

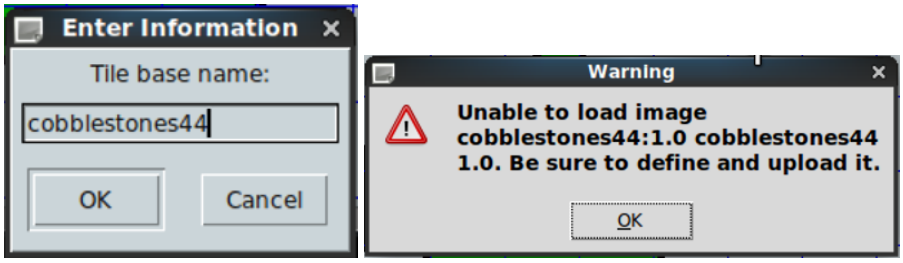




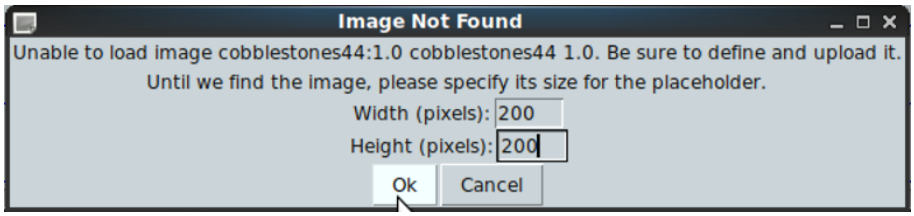


The stamp tool, which is used for laying down image tiles (usually for textured backgrounds to use for floors and walls, but could conceivably be used for anything desired). Its use is thoroughly covered starting on page 82 in Chapter 5.

However, there is one additional thing to note. If your map client does not yet know where to find the requested image (perhaps because you haven't yet loaded it up to the server or loaded the image definition map file—`tiles.map` in the examples in Chapter 5), it will temporarily display a placeholder image on the map to allow you to continue working. When the mapper has a real image to load, it will display that instead of the placeholder.



It will prompt you for the image dimensions so that it can make the placeholder box the same size as the image, to make it easier for you to work with the rest of the map you're building.



Colors, Snap, and Thickness

The next group of toolbar buttons affect how the objects you create with the other tools are drawn on the map canvas.



This control is a toggle; it is activated when pushed, and deactivated when pushed again. When activated, shapes will not be filled. For example, a rectangle will only be drawn as the outline of its perimeter rather than also filling in its interior with a solid color.



Clicking on this button brings up a color selection dialog to choose the object fill color. This is used for interior fills. Note that line objects are drawn with the fill color, not the outline color.



This brings up a color selection dialog to choose the color used for drawing the perimeter outlines for shapes except for line objects, which use the fill color instead.

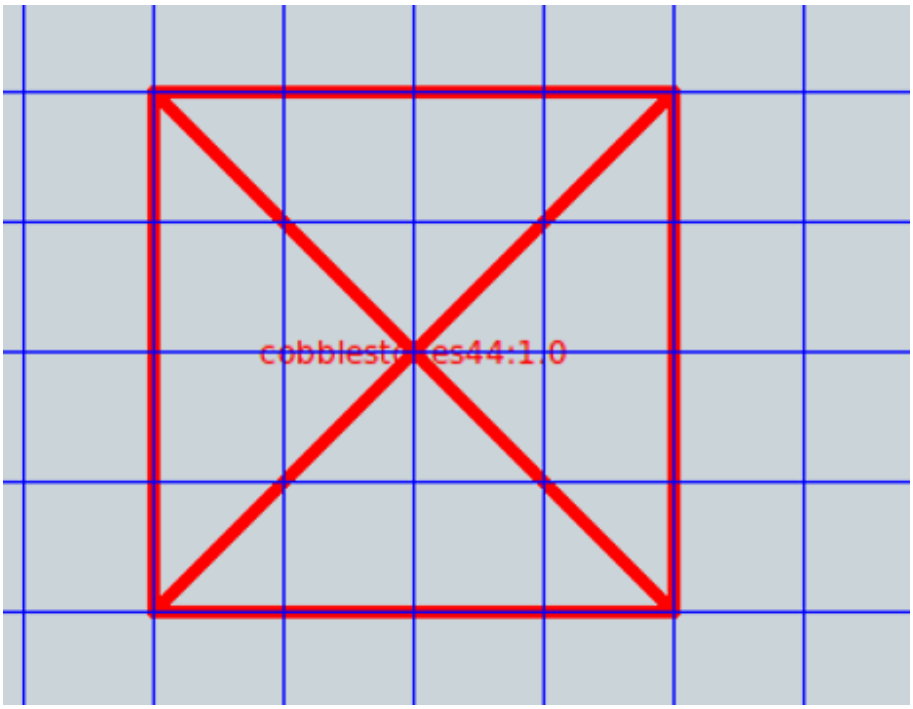


Figure 4.2: Image Placeholder



Clicking on this button cycles between a number of fill patterns. If you selected the no-fill button (see above), then no fill pattern will be used regardless of this button's setting. Otherwise, shapes will be filled with a solid color if no pattern is selected (indicated by an "X" in this button), or with a stipple pattern you selected otherwise, drawn in the selected fill color. Any open space around the fill pattern's pixels will show through to whatever is under the shape. As of this writing, the set of fill patterns includes 12%, 25%, 50% and 75% density shading patterns, but more may be available in future releases.



Clicking this button cycles through the various grid snap settings, which include:

-  No grid snap; objects appear exactly where clicked.
-  Full grid snap; points will be on grid intersections.
-  1/2 grid snap; points will be on grid intersections or half-way between them.
-  1/3 grid snap; points will be on grid intersections or each third of the way between them.

 1/4 grid snap; points will be on grid intersections or each fourth of the way between them.

 Clicking this button cycles through the various line thickness settings, which affect the drawing of line objects and the lines around the perimeters of other objects, in 10 steps from  to  .

Object Removal

These buttons remove objects *en masse*.

 These are general deletion buttons. Clicking  removes all map elements other than creatures, while  just clears the creature tokens.

Miscellaneous Controls

The last group of controls on the toolbar adjust your view by zooming in and out, and loading and saving map files.

 These buttons, respectively, zoom in so everything on the map is twice the size, zoom out so everything is half the size, and resets the zoom to the default view.

 Each of these select an existing map file to load or unload into the mapper. Clicking  loads a map file's contents into the mapper, replacing all existing content. By contrast, clicking  does the same but merges its contents with the existing objects on the map.

The  button unloads content from the map. Specifically, it opens the file just as if loading the content, but for every object saved to that file, the mapper deletes that object from its canvas if it was there. It's not an error if some of the file's contents weren't already loaded.

 This prompts you to specify a file into which the mapper will save the contents of its canvas. All of the objects, including PCs, NPCs, and monsters, are saved so that the same contents will be on the map when the file is reloaded later.

 Exits the mapper program.

The other tools will be explained when we get to their functions, including multi-user setup.

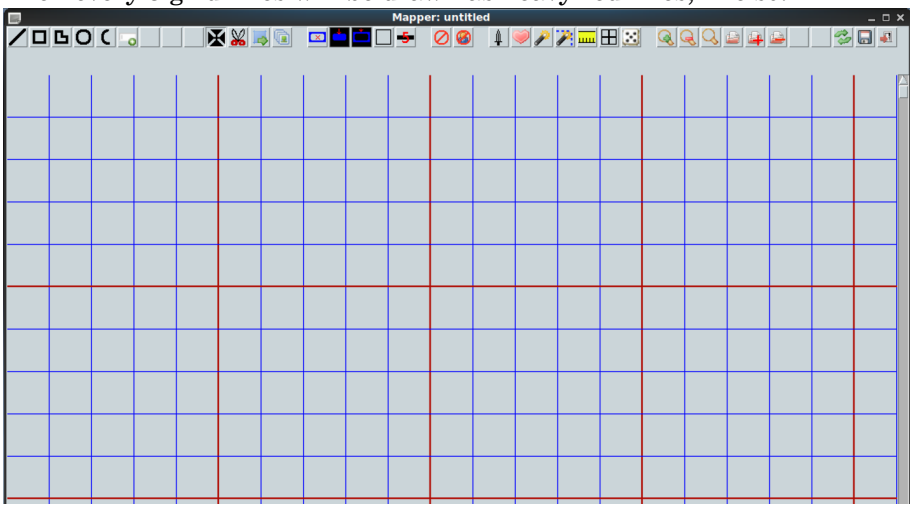
Major and Minor Guide Lines

Sometimes it is convenient to have additional grid lines to help you compose a map, especially if you're going from another larger map for reference. When you launch the mapper too, you can supply a command-line switch to

enable heavy lines at certain intervals. For example, if we run the mapper as

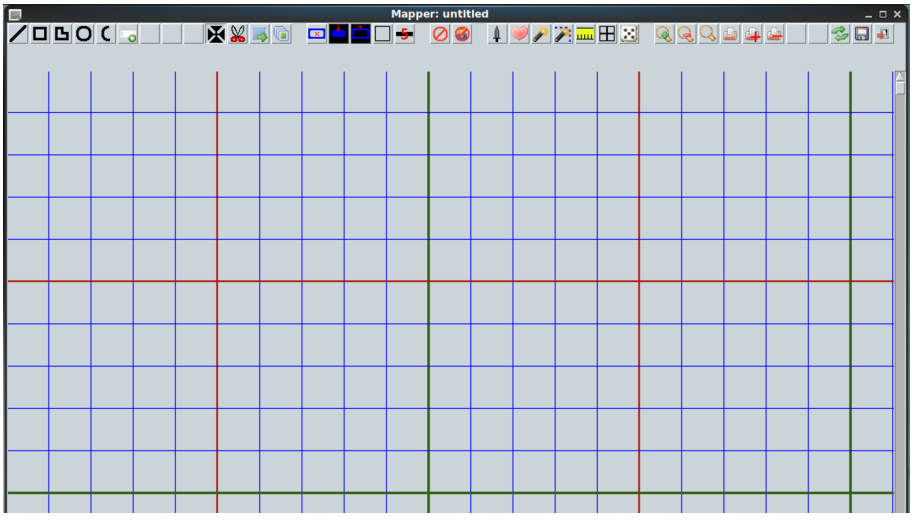
```
$ gma mapper -- --guide 5
```

Then every 5 grid lines will be drawn as heavy red lines, like so:



We can go further and have a more major set of lines, this time green, at every 10 lines:

```
$ gma mapper -- --guide 5 --major 10
```



To make these permanent we can set them in the mapper's preferences. From the main menu, choose Edit ⟩ Preferences... as described fully starting

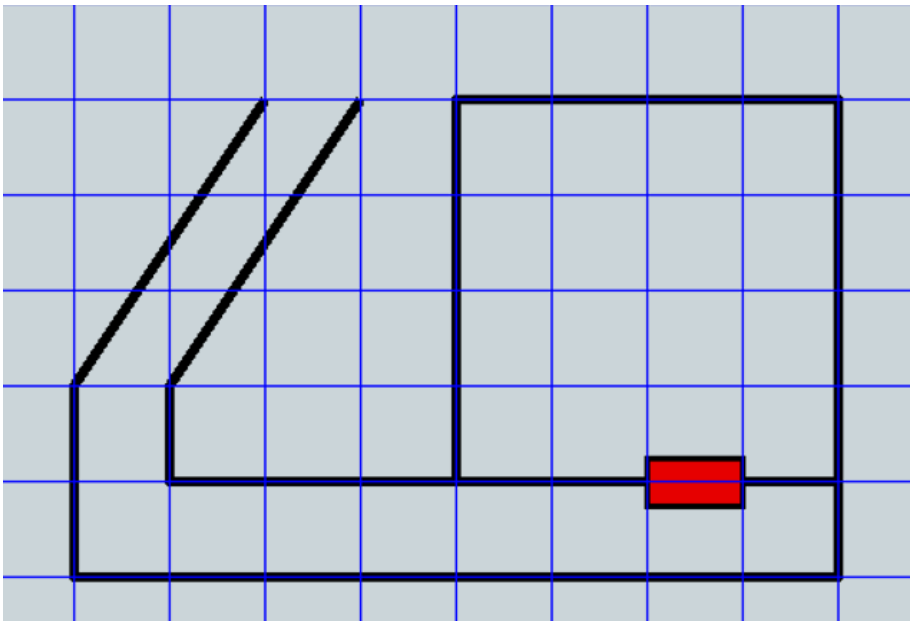
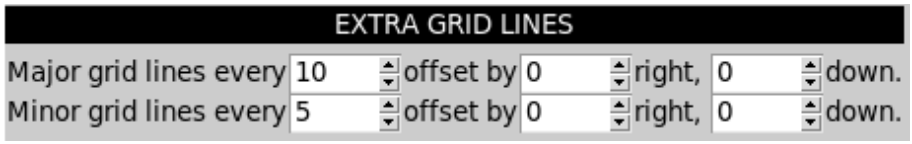


Figure 4.3: Simple Dungeon Map Sketch

on page 726. On the “Appearance” tab, fill in these values in the “Extra Grid Lines” section.

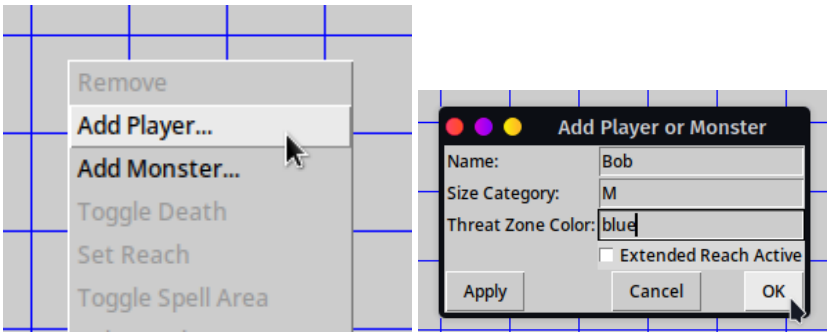


A Simple Dungeon Map

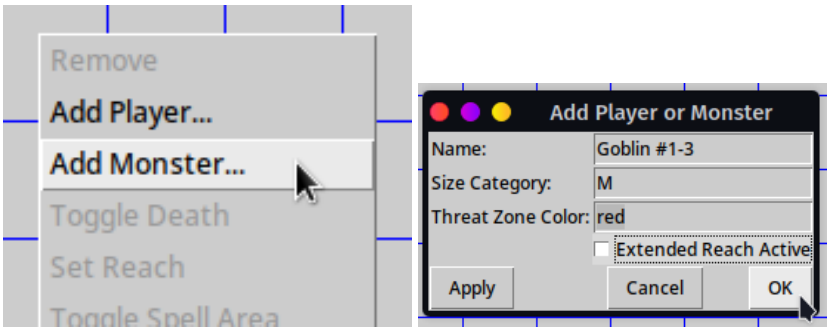
Using a few basic shapes we can sketch out a simple floor-plan of a dungeon in a few seconds. This is handy if we need to draw a quick map on the fly during a game. An example is shown in Figure 4.3. This was made with a filled rectangle for the door and a few lines.

Using Maps During Combat

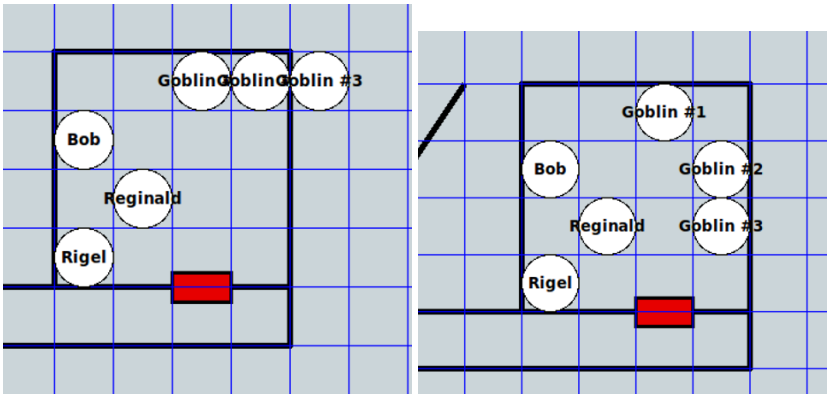
Let’s return to the combat example from Chapter 3. Suppose the party has walked into this dungeon room and encountered the three goblins. After quickly sketching out the room, the we place the characters on the map, starting with Bob



and similarly for Reg and Rigel. Now to add the goblins. We don't need to enter each goblin separately since they are named "Goblin #1," "Goblin #2," and "Goblin #3." We can simply enter the name pattern "`Goblin #1-3`" in the dialog:



This will place three goblin tokens in a row on the map, which we'll just arrange into position as desired.



When we go into combat, the mapper can help. Click the Combat button ( ) on the toolbar. The map now shows the threatened spaces around each combatant. To avoid confusion, we'll look at just two combatants here:

As medium-size creatures, both Bob and Goblin #1 threaten the spaces immediately adjacent to them, as shown by the shaded zone around each. Since neither is in the other's threat zone, neither is in range for *mêlée*.

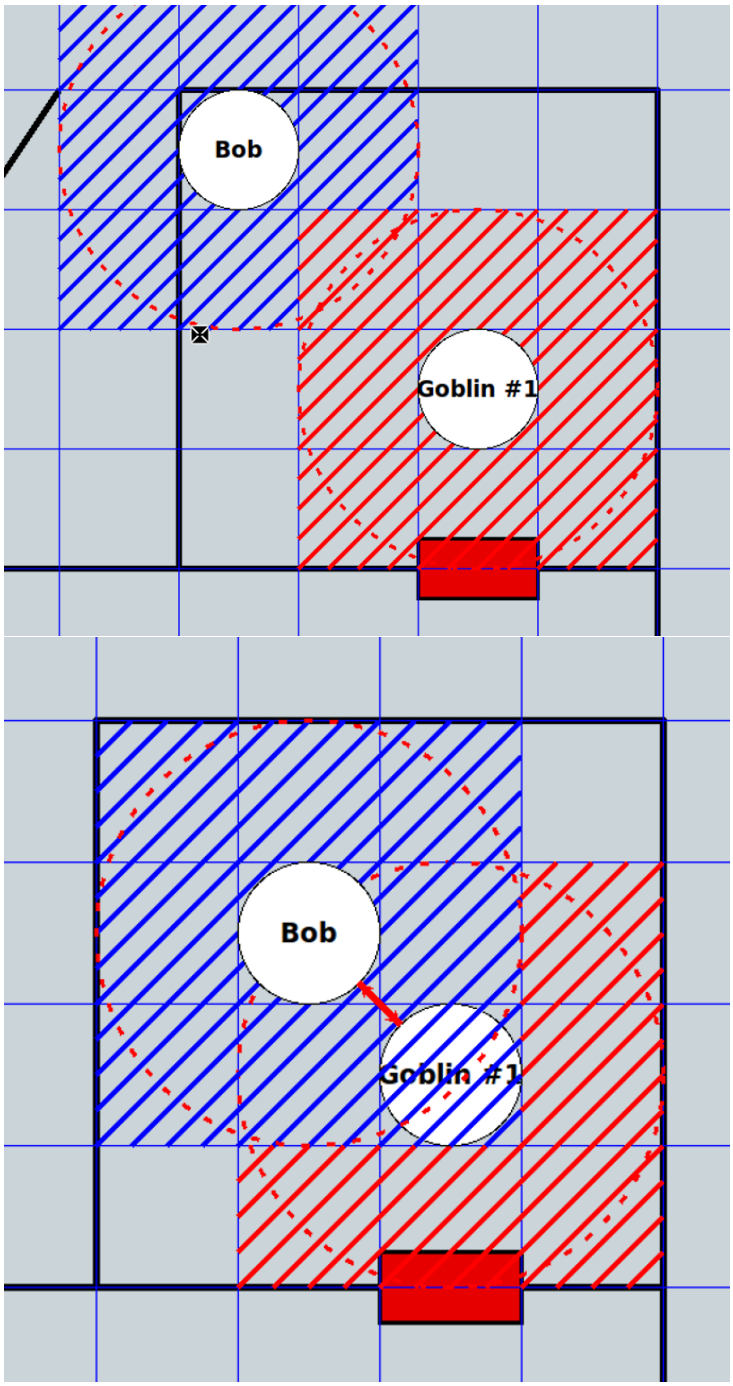


Figure 4.4: Bob vs. Goblin

One five-foot step later, however, and we have an exciting combat underway. In addition to the shaded zones, the mapper draws arrows from each creature to the creature(s) it threatens.

Spell Areas

In Appendix I we saw how to turn on a personal spell area of effect that emanates from a creature token, and moves with them. Here, we'll show how to mark fixed areas on the map as being within a spell's area of effect.

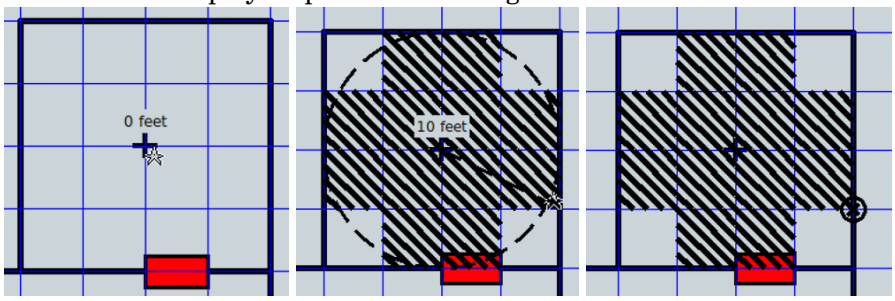
The zone is marked using the current fill color, so start by clicking on the fill color button ( ) to choose the desired color (and make sure you don't have filling turned off). Then click the Spell Area button ( ) on the toolbar.

With this tool selected, an option button is available to cycle through the different spell area of effect shapes:

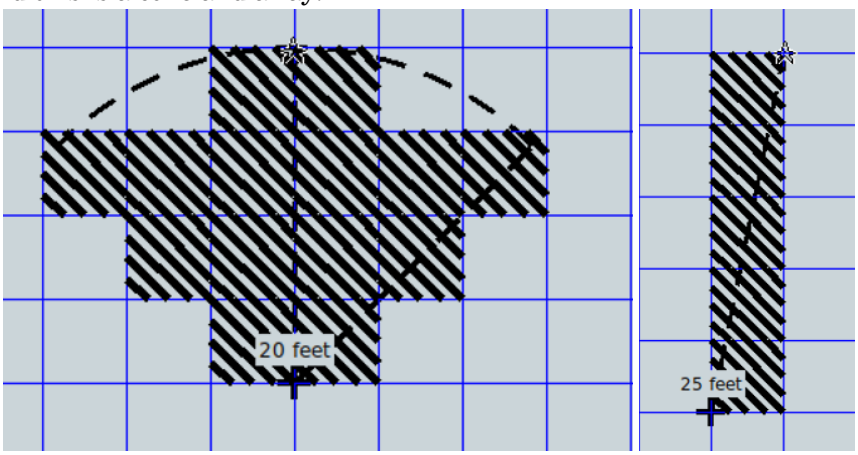
- Circular area to a specified radius.
- Cone spreading out from a point.
- Straight-line ray.

In each case, click on the point where the spell originates (usually the intersection of squares), then move the mouse to the edge of the area of effect. You will see the affected squares highlighted as you move the mouse. When satisfied, click again on the end point to commit the area of effect object to the map.

This is the step-by-step view of drawing a radius effect:



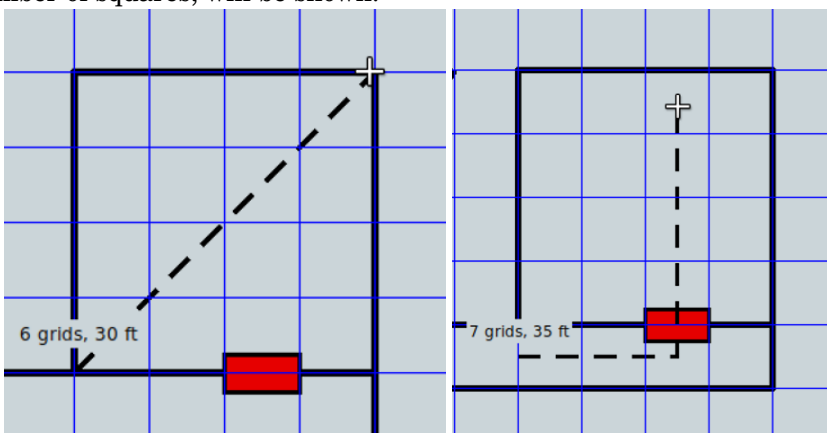
And this is a cone and a ray:



Once placed, this is a map object like any other shape; it can be removed by using the delete tool ( ).

Measuring Distances

The ruler tool ( ) gives you a way to measure distances along a path. Just like with using the line-drawing tool, click on the starting point of the path, then click on the end point. Or, continue clicking from point to point along a connected path of line segments. The total distance, rounded to an even number of squares, will be shown.

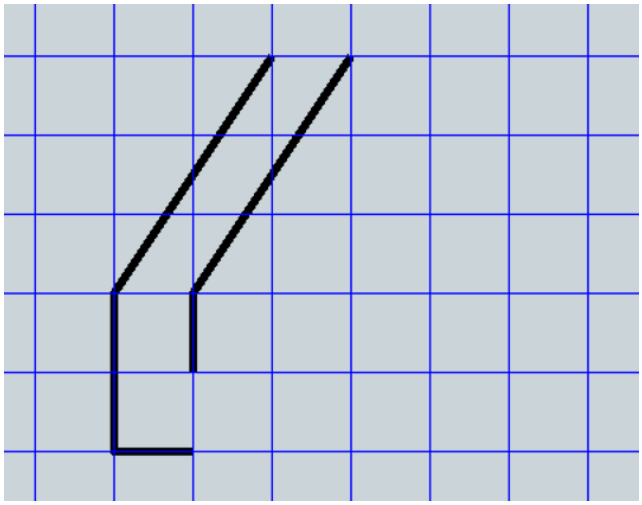


Mapping Piecemeal

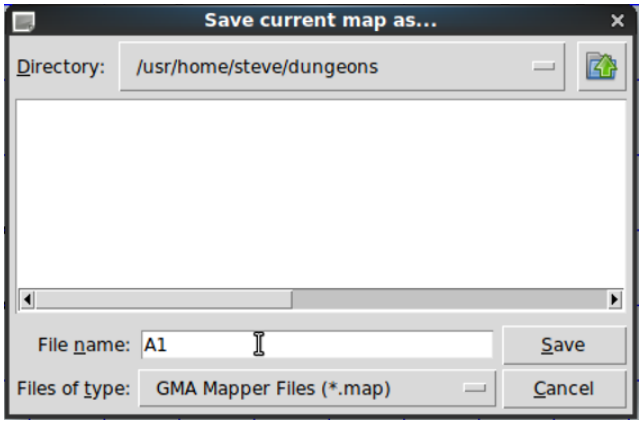
For a real game, you would want to prepare a set of map files in advance. Ideally, you'd make a separate map file for each room or discrete area the party will encounter, so you can reveal the map to them incrementally as

they explore. (We're assuming the players can see the map somehow, either by projecting it on a big monitor, or by setting up a server so they can view their own connected clients.)

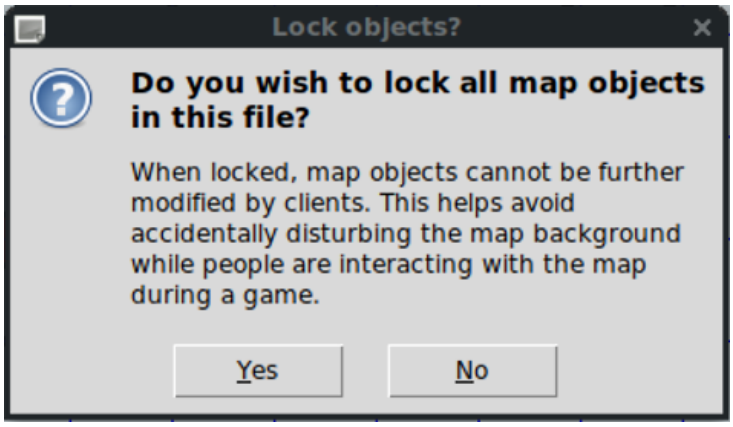
Let's set up our simple example as a multi-file dungeon map. We'll start by drawing the diagonal corridor piece.



Save this to a file by clicking the Save icon () on the toolbar. If this is area A1 on our map, we'll just save it to a file with that name.

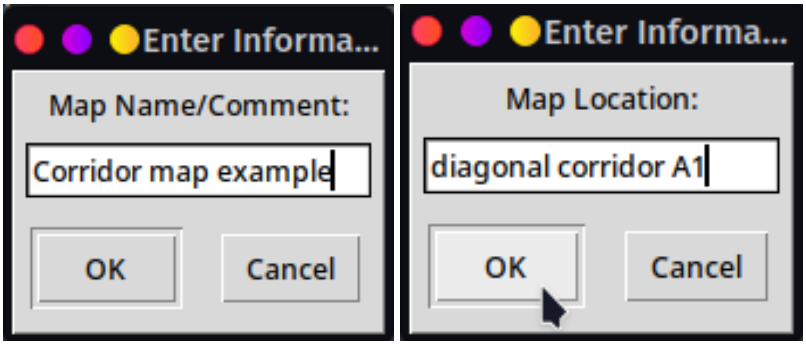


When we save the file, the mapper asks us if we want to lock the objects in the file.



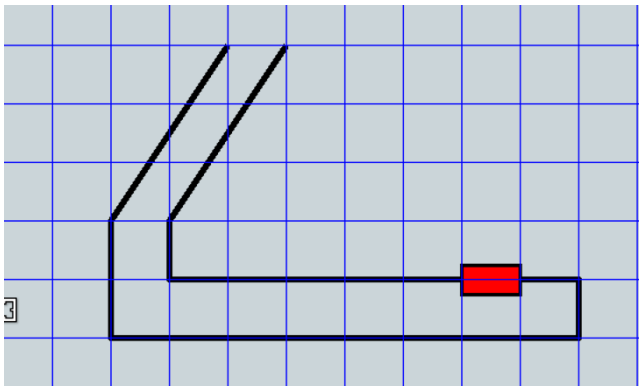
This is a good idea to do if we're finished setting it up and it's ready to use for our game. The locked objects can't be deleted or moved around so we can avoid game-time mishaps that destroy the map due to a misplaced click.[2]

We're also prompted to provide a descriptive comment and for the location name the map represents. These will be saved to the file along with the map data.

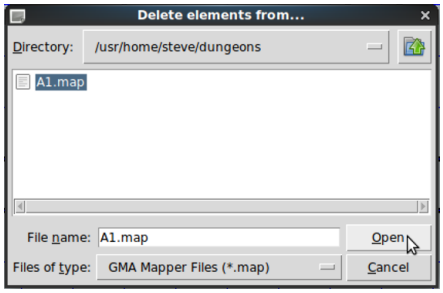


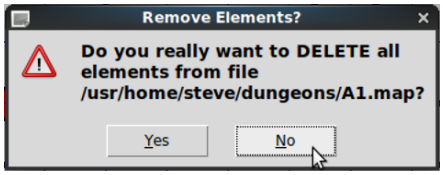
Now we'll add the rest of the corridor and the door to the chamber.

[2]This is especially true when network connections are set up to allow players to run mapper clients of their own, too. That just multiplies the chances of someone accidentally mucking up the map.

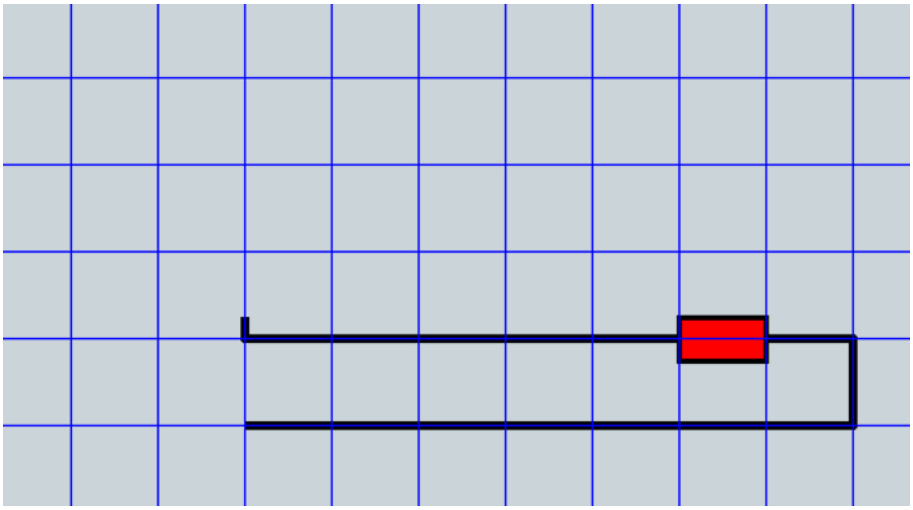


We will save this to its own file (A2), but we don't want to include all of the content from corridor A1 since that's already saved to its own file. So we will *unload* everything that was saved to the A1 file first, by clicking on the Unload icon () and then selecting the file `A1.map`, and clicking Yes on the dialog that asks us to confirm our intentions in that regard.





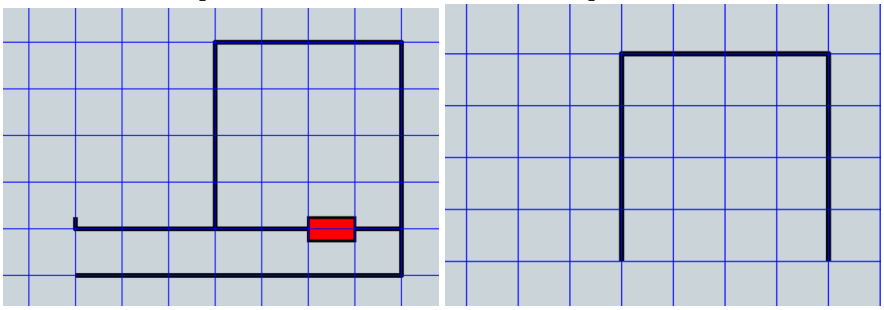
Now we only have area A2 on the canvas.



We'll save that to the file `A2.map`.

Next, we draw the chamber on the other side of the door, unload the

contents of `A2.map`, then save what's left to `A3.map`.



Revealing the Map Incrementally

As the party explores the dungeon, we will reveal the map as they go, since we have each piece stored to its own map file. First, when they enter the corridor A1, we load `A1.map` by clicking the Load button ( ) and selecting that file from the list. As they turn the corner into area A2, we merge the contents of file `A2.map` by clicking the Merge button ( ). And finally, when they open the door, we load the chamber by merging file `A3.map`. Figure 4.5 shows the step-by-step progression the players will see.

Offline Manipulation of Map Files

Once you have a collection of map files like the `A1.map`, `A2.map`, and `A3.map` we created above, you can reload them into the mapper, make changes to them, and save them again.

Oh, wait. You can't do that if you locked all of the objects when you saved them. (Although you can add more new objects to the files.)

Never fear, dear reader! There are a few command-line utilities that will manipulate the contents of map files independently of the mapper program. To the current point, running the command

```
$ gma map-lock --unlock A[123].map
```

will unlock everything in those three files again. Now we can load them into mapper and edit all of their contents again. We can also run

```
$ gma map-lock A[123].map
```

to lock all of the files' contents again.

After a while we realize that when we made the maps for areas A1, A2, and A3, we didn't leave enough room for more areas that are to the left of them on the map. It would be a pain to edit every one of those files and

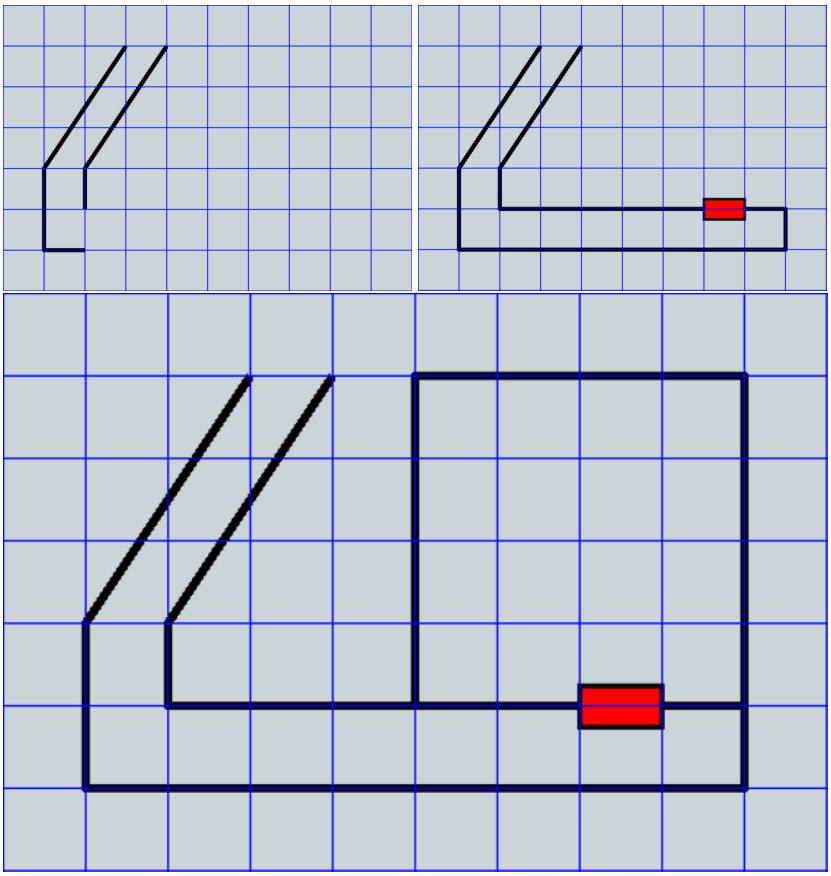


Figure 4.5: Incremental Reveal of the Map

move their objects with the mouse, one at a time, hoping we did an accurate job of all of that.

Again, never fear! The `map-translate` program comes to our rescue. If we decide that we need to just move everything in all of our `.map` files 500 pixels to the right, we can simply run the command

```
$ gma map-translate -x 500 *.map
```

The details for these and other command-line utilities are documented in Appendix D.

CHAPTER 5

LEVELING UP: ADDING GRAPHICS TO YOUR MAPS

> The difference between the quest for the Holy Grail and someone saying "bring me a cup" is the flavor text and the number of stops involved.
>
> —Bryan Fields
> *Life With a Fire-Breathing Girlfriend*

THE FOLLOWING TUTORIAL describes how to work with image files and the GMA mapper tool, to create creature tokens, background tiles, and whole-room maps from prepared images. It is best if you read it from start to end, since each section builds on the previous one.

How To Use Creature Tokens

By default, the mapper will create simple creature tokens for the players, their allies, and the creatures they encounter. These are circles with the name of the creature written across the center of the circle. When the creature dies, the circle inverts colors so it is solid black with white lettering.

It's a lot more interesting, however, if the token is an image of the creature in question. In order to do this, you'll need to prepare an image file which we will adapt for use with the mapper tool. The image can be in any

format[1] and any size. For best results, it should be an image format which supports alpha channels (i.e. transparent pixels)—usually GIF or PNG—and the higher the original image resolution the better.

For our example, we will use the image shown in Figure 5.1 for a wizard character named Agnes.[2]

This file happens to be in JPEG format, but we will use GMA tools to convert it to PNG so that we can get all the features we need (and because the GMA Mapper doesn't support JPEG images).

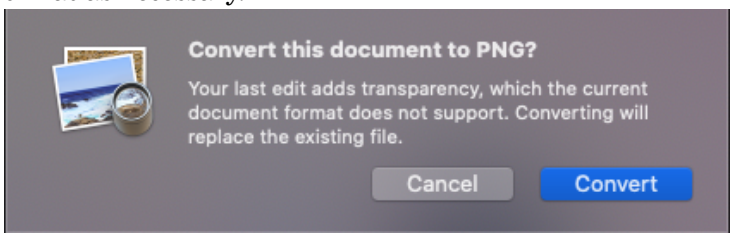
Wonderful image though this is, there's far too much of it to fit within a creature token, especially at the sizes the Mapper typically displays, so we will need to crop it down. For medium-size creatures and smaller, that usually means we have about enough room to show their head. For larger creatures, we might be able to fit more of the creature in the token, depending on its shape.

Also, although image files generally used by the mapper are rectangular, those used specifically for creature tokens are *square*. We will need to crop the image to a circular shape that exactly fits into a rectangular image space.[3]

I edited this using the Preview tool that comes built-in to macOS, for the sake of demonstrating how this can be done without needing more powerful third-party tools, but tools such as GIMP, Photoshop, etc., work nicely as well.

With the image loaded in your editing program (Preview in this example), select the circular selection tool and outline the desired area, taking care to keep the selection area a circle and not a non-circular ellipse (see Figure 5.2).

As expected, the Preview program prompts us to convert the image from JPEG to PNG format, since we need a format that supports transparency. Depending on the tool you're using, you may need to manually convert the image format as necessary.



The cropped image should look like Figure 5.3. This is sufficient, but if you want to go the extra mile, it is better to remove the background be-

[1] Any format known to ImageMagick, at least.

[2] In honor of Agnes Waterhouse, the first woman executed for witchcraft in England, in 1566.

[3] This all means that while we crop the visible part of the image in a circular area, the image is actually square, with the parts outside the circle set to transparent so whatever is behind the creature token shows through. This makes the image appear to be circular for all intents and purposes on the mapper.



Figure 5.1: Original Image Source for Agnes
Art credit: Igor Esaulov

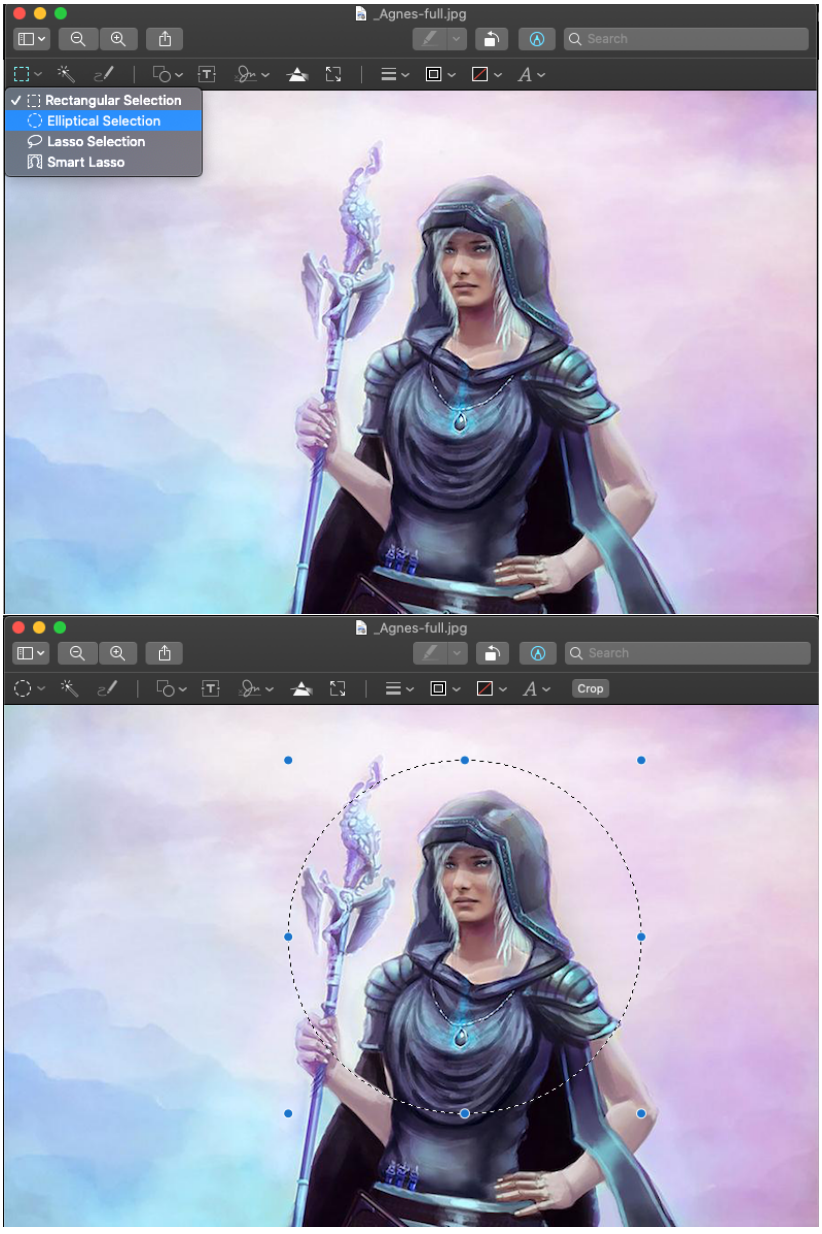


Figure 5.2: Circular Crop for Token Image

hind the character, so the image is just their face (or whatever part of their body is showing in the cropped image). This way, the mapper's token background coloring will show through. For example, when the creature dies, the mapper token circle will flip from white to black, which will be more dramatic if you can actually see it through the graphic image, and makes for a more visually distinct indicator of the creature's status.

Now we save this to a file called `#Agnes.png`.[4]

We now want to make a version of the image which will be displayed when the creature is killed. For my game, I just desaturate the image to grayscale, but for more cartoonish images you can put Xs through their eyes, flip them upside-down, or whatever you like. Save this version of the creature token image as a file named `%Agnes.png`.[5]

If you *only* want to desaturate the colors of the image, you don't need to make a separate version of the image now at all. Just run `rendersizes` with the `-k` option to automatically generate the "killed" version of the token for you. We will show an example of this shortly.

The mapper will simply display the image as-is at the coordinates where the creature is located, so the image itself needs to be scaled to the same size as the other map elements. If the map is zoomed, we need to use a version of the image scaled to the new display size for that zoom factor. Since the mapper itself can't rescale images on the fly, we need to pre-generate all of the different sizes of all our images for the mapper to use as needed. All of those things are done for us by the `rendersizes` script:

```
$ gma rendersizes -f tiles.map M [#%]Agnes.png
Converting images to M-size image tiles into tiles.map...
#Agnes.png: 12 25 50 100 150 200 300 400 600 800 1600
%Agnes.png: 12 25 50 100 150 200 300 400 600 800 1600
Fixing EOF in tiles.map
```

This creates scaled versions of both image files at eleven different sizes, to facilitate the display of the creature token at a few different zoom levels and allowing for the creature to alter its size at those zoom levels. These are written to files `#Agnes@50.png`, `#Agnes@100.png`, etc., which the mapper will expect to read when displaying a creature called `Agnes`.

If you want the "killed" version of the token to be generated automatically for you, that command would be:

---

[4] You may alternatively name the file `^0^Agnes.png` if that's more convenient to type or is more consistent with other images you're using, as explained below. The Mapper considers both filenames to refer to the token for a creature named "Agnes".

[5] Alternative names which the Mapper will also recognize for the "killed" version of Agnes's token include `%^0^Agnes.png` and `%#Agnes.png`.

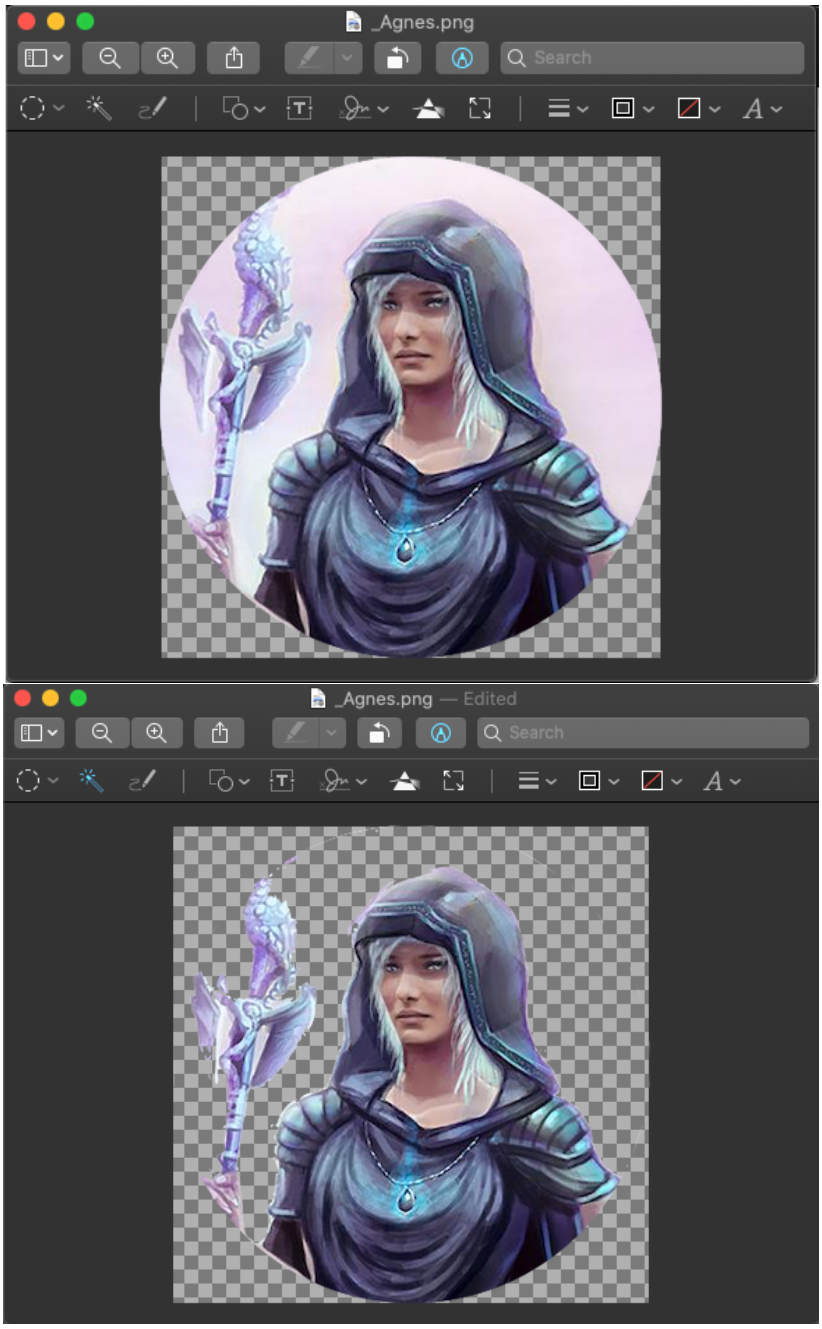


Figure 5.3: Cropped Image With and Without Background

```
$ gma rendersizes -k -f tiles.map M '#Agnes.png'
Converting images to M-size image tiles into tiles.map...
#Agnes.png: 12 25 50 100 150 200 300 400 600 800 1600
%Agnes.png: 12 25 50 100 150 200 300 400 600 800 1600 (genera
ted)
Fixing EOF in tiles.map
```

It also appends data to the `tiles.map` file, which will look like the example in Figure 5.4.

Note that `rendersizes` output the images in PNG format by default, but if you give it the `-g` option it will instead produce GIF files. The mapper expects to read PNG by default, but it can be set to read GIF files by giving it the `--image-format gif` option or setting the approprite preference setting inside the mapper.

See the manpage for `rendersizes`(6) on page 532 for full details about how to use this with different sizes of creatures and other options.

Displaying Creature Images On the Map

At the start of play, load `tiles.map` into the mapper as if it were a map room. Although this particular map file doesn't actually contain any visible map elements, it does define the image names by which the mapper will know them, and informs the mapper of the file pathname where it may find the images themselves. Note that this means you need to run the mapper from a directory where the pathnames as stored in `tiles.map` make sense so the mapper can actually find the image files.

You can add the `-L` option to `rendersizes` to make this process a little easier. This will cause `rendersizes` to go ahead and copy the image files directly into your mapper's image cache so they're ready to go immediately. As we'll see in Chapter 11, when we're preparing a map for use in a multi-user environment where the image files are served over the network, using `-L` saves you a lot of needless work during map preparation. We will henceforth just assume `-L` will be used wherever applicable for the examples that follow.

Now, *voilà!* the graphical token for Agnes will show up on our mapper (Figure 5.5).

Preparing Images for Server Distribution

If you are using the full server-based mapper arrangement as we'll discuss in Chapter 11, you'll want to use the `-w` option to prepare the files for uploading to a web server:

```
__MAPPER__:22
«__META__» {
   "Comment": "Generated from GMA rendersizes script",
   "Timestamp": 1650611775,
   "DateTime": "Fri Apr 22 00:16:15 PDT 2022"
}
«IMG» {
   "Name": "#Agnes",
   "Sizes": [
     {"File": "#Agnes@12.5.png", "IsLocalFile": true,
      "Zoom": 0.25},
     {"File": "#Agnes@25.png", "IsLocalFile": true,
      "Zoom": 0.5},
     {"File": "#Agnes@50.png", "IsLocalFile": true,
      "Zoom": 1.0},
     {"File": "#Agnes@100.png", "IsLocalFile": true,
      "Zoom": 2.0},
     {"File": "#Agnes@200.png", "IsLocalFile": true,
      "Zoom": 4.0}
   ]
}
«IMG» {
   "Name": "%Agnes",
   "Sizes": [
     {"File": "%Agnes@12.5.png", "IsLocalFile": true,
      "Zoom": 0.25},
     {"File": "%Agnes@25.png", "IsLocalFile": true,
      "Zoom": 0.5},
     {"File": "%Agnes@50.png", "IsLocalFile": true,
      "Zoom": 1.0},
     {"File": "%Agnes@100.png", "IsLocalFile": true,
      "Zoom": 2.0},
     {"File": "%Agnes@200.png", "IsLocalFile": true,
      "Zoom": 4.0}
   ]
}
«__EOF__»
```

Figure 5.4: Example of Generated Mapper File with Images

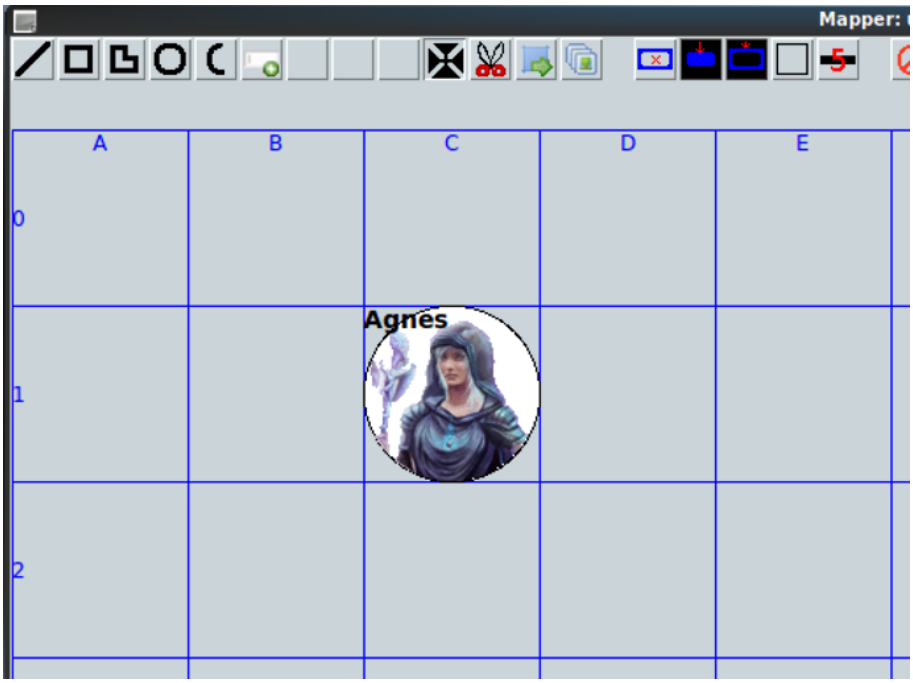


Figure 5.5: Finished Character Token On the Map

```
$ gma rendersizes -wL -f tiles.map M [#%]Agnes.png
Converting images to M-size image tiles into tiles.map...
#Agnes.png: 12 25 50 100 150 200 300 400 600 800 1600
%Agnes.png: 12 25 50 100 150 200 300 400 600 800 1600
```

The full details behind this use case are found in Chapter 11.

Semi-Transparent Creature Tokens

To take full advantage of the mapper's features, you'll also want to generate a set of semi-transparent versions of all the creature token images. The `rendersizes` script can do that automatically via the `-t` option. Thus, the typical `rendersizes` script invocation to prepare a full set of creature token images would be:

```
$ gma rendersizes -Lwkt -f tiles.map M ./#Agnes.png
```

How To Use Map Tiles

Building on the previous set of concepts, we will now see how to use images as *tiles* on the map itself, to provide textured floors or whatever we like. The principle here is the same as with the creature tokens, but now we can use arbitrary rectangles instead of squares.

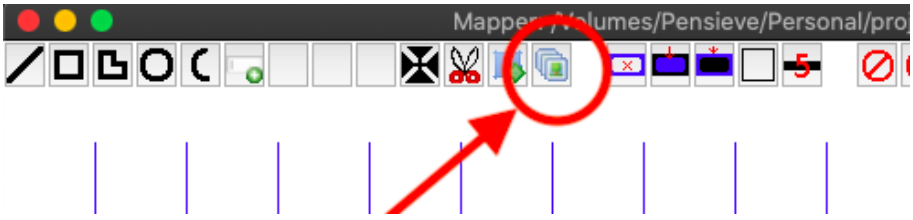
Suppose we want to put a cobblestone pattern on the ground. We find a pattern that looks right and save it as, say, `cobbles44.jpeg`. We intend this to be sized to fill a 4×4 square area (20×20 feet).

We use the `rendersizes` script to scale the image so that it just fits in a 4×4 grid area on the map, producing versions sized for the standard mapper display plus two zoom levels in and out from that base size.

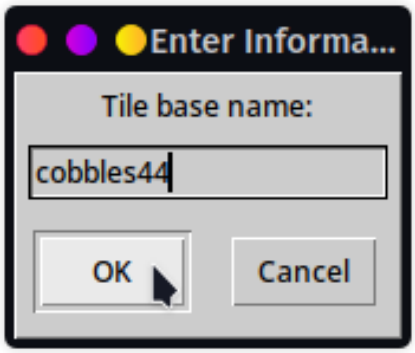
```
$ gma rendersizes -L -f tiles.map grid 4 4 cobbles44.jpeg
Converting images to 4x4-size image tiles into tiles.map...
cobbles44.jpeg: 50x50 100x100 200x200 400x400 800x800
```

Loading the `tiles.map` file informs the mapper that the image exists and where to find the file(s) to display for it. (Or if you use the `-L` option, `rendersizes` will copy the files into the mapper's cache so they're immediately usable.) Having done that, we can "stamp" this image onto the map using the stamp tool. Since we sized it to be an even number of squares across and down, it's probably best to have grid snap enabled on the mapper, but that is not required.

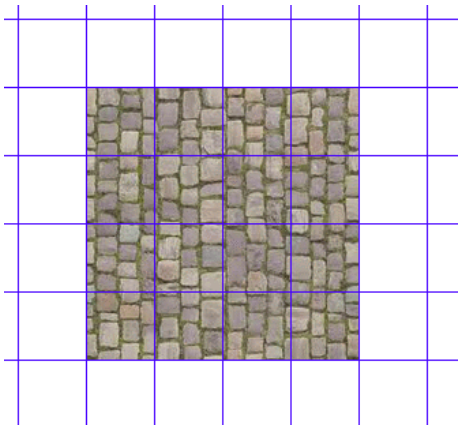
On the mapper, select the stamp tool.



To change the stamp tool's current image, right-click, then type in the image file name (just the image name, without its directory or extension/suffix).



Now left-click where you want the upper-left corner of the image to go on the map.



Click again where you want another, and again, and again as desired. You can add other map features on top of the tiles.

Full Dungeon Map Images

If you wish to go all the way and just use dungeon map images directly from modules or other sources on the map rather than hand-drawing all the walls, doors, furniture, and so forth, this is easily arranged using a variation on the concepts we've already covered above. The tricky part is getting the image scaled so that the squares on the original image match up with the grid lines on our map tool. Fortunately, `rendersizes` comes to the rescue here as well.

Let's take a simple prepared floorplan of a dungeon, such as a few rooms from the classic D&D module B2, *The Keep on the Borderlands*. This particular map image was created by Mike Weem, as a nice reimagining of the venerable Caves of Chaos dungeon setting from that module.[6] (see Fig-

[6] He also generously released it to be used and enjoyed freely.

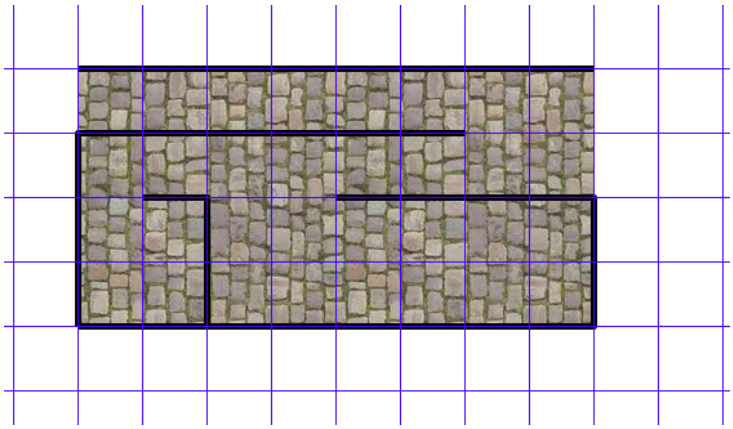


Figure 5.6: Stamped Image With Walls Drawn On Them

ure 5.7)

Looking at the GM copy of the map, we see there are secret doors the players shouldn't see yet, but fortunately we have a player version of the map. We'll resize that as a tile to display in our mapper tool.

We want to run it through `rendersizes` but what scale factor to use? How many squares across and down is it, considering that the image extends some unknown amount beyond the grids of the dungeon rooms? We need a way to measure the scale of the map image and match that to our own. Using a graphics tool of your choice (again, I used the macOS Preview program), measure some distance on the original file that you know corresponds to a known number of 5-foot squares. The easiest way to do this is to measure the distance across as many contiguous squares on the original images as you can, and see how many pixels that is.

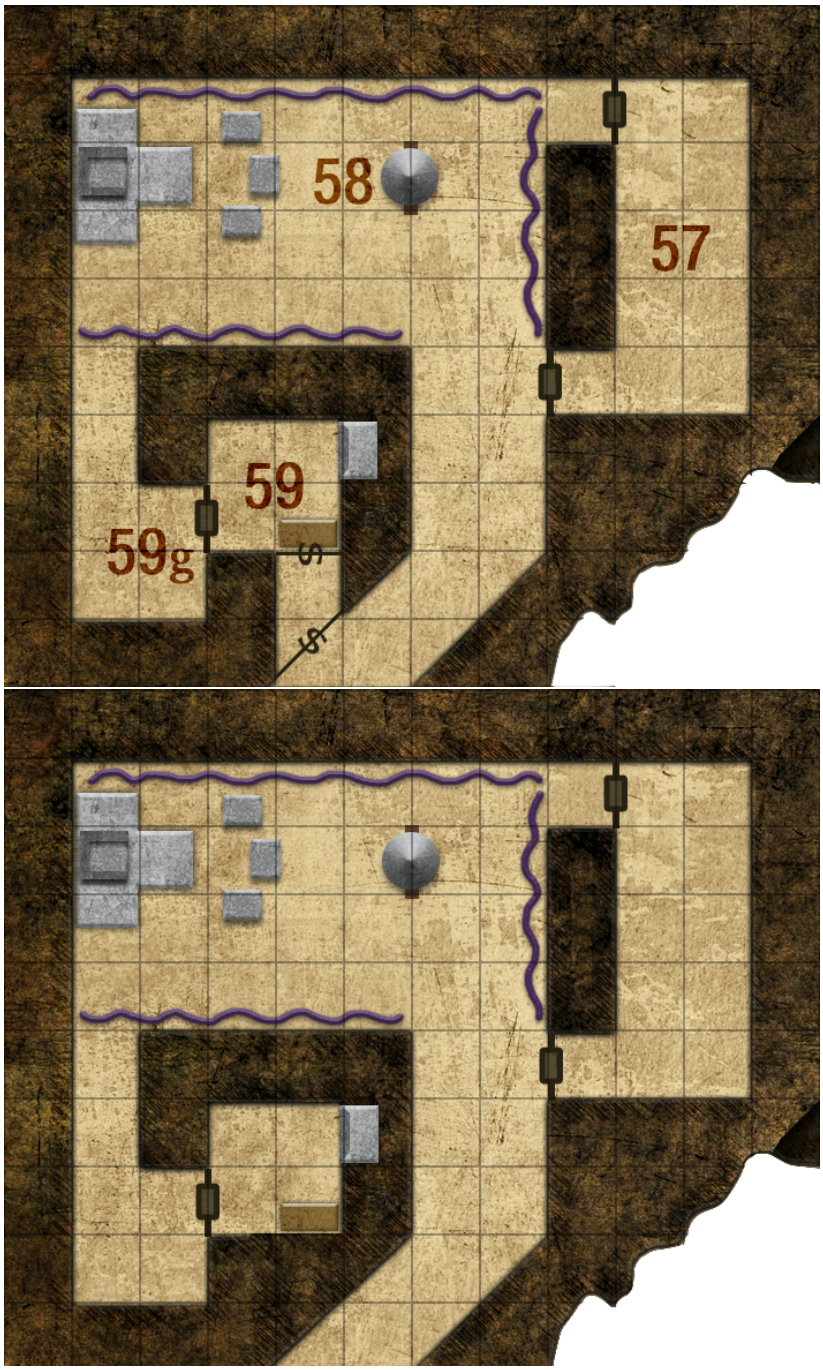
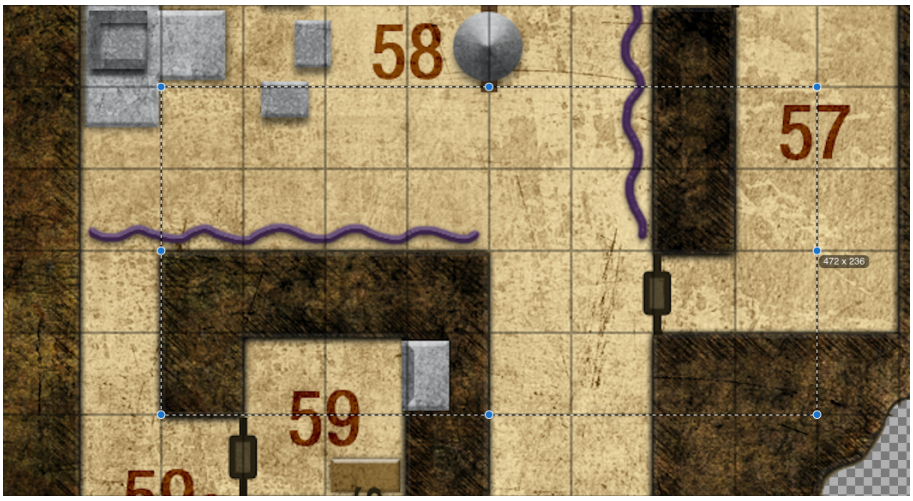


Figure 5.7: Dungeon Rooms From Module B2 (GM and Player Versions)



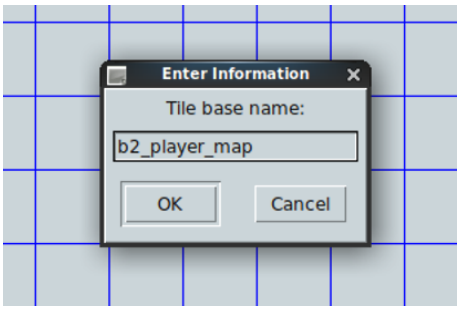
Here we see that 472 pixels corresponds to 16 squares on the map. (The squares as drawn on the module's map are 10 feet across, but the mapper tool uses 5-foot squares.) With that information, we can get `rendersizes` to do the rest for us:

```
$ gma rendersizes -L -M B2 -f tiles.map scale 472 16 \
b2_player_map.png
Converting images to scale-size image tiles into tiles.map...
b2_player_map.png: 12 [40.6779661016949%] 25 [84.745762711864
4%] 50 [169.491525423729%] 100 [338.983050847458%] 200 [677.9
66101694915%]
```

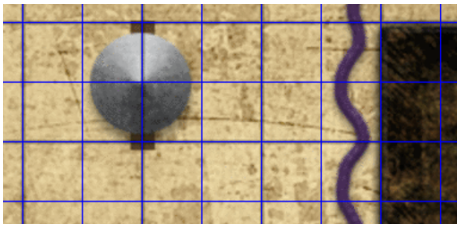
Now we have a tile we can put on the map. In this case, you'll probably want to leave grid snap off, and use the Object Move tool ( ) to accurately position the tile so the grid lines match up exactly. When using the Object Move tool, once you have started moving the image with the mouse you can make fine adjustments as needed using the keyboard. The arrow keys ↑, ↓, ←, and → (and respectively the keys k, j, h, and l), will nudge the image one pixel at a time in the desired direction. Turning on and off the grid display can help you verify that you have the image aligned precisely under the mapper's grid lines.

In this mode, we can also use the keys u and d to move the object up and down in the on-screen stacking order (also known as the object's *z* coordinate), or the keys f and b to move the object all the way to the front or back of the stack of objects.

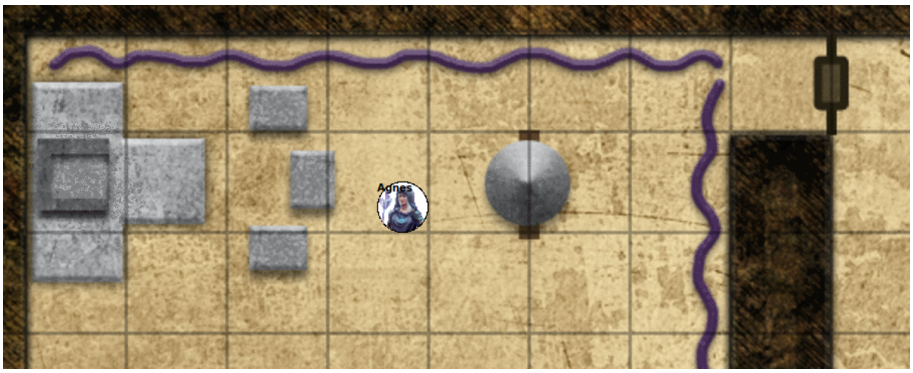
Start the mapper and load `tiles.map`, then select the Stamp Tool ( ), right click, and enter the tile image name.



Stamp down the image and move with the Object Move tool into the desired position, so the mapper's grid lines align with the ones on the map image (or if the image doesn't have lines, the rooms line up with the grids as they should).



If you like, you can switch off the mapper's built-in grid using the grid toggle button (⊞) and just use the grid lines already in the prepared tile image. Here's our map with the grids off and our wizard friend waiting in the room:



Save your map to a file so you can load it up later during game play. The map file will contain a reference to the tile image we just imported from the prepared map, along with whatever other objects you add to it.

If the tiles are huge enough you don't want to deal with the biggest zoom level, use the `-S` option to `rendersizes` to suppress generating the image at that magnification factor. Likewise, the `-s` option suppresses the two largest sizes.

Piecemeal Dungeon Map Images

As we discussed in Chapter 4, we often want to reveal the map one area at a time as the party explores the dungeon, so we don't spoil any surprises too soon. We will take the same approach we did before, creating a separate `.map` file for each area, but since we're now doing this with graphic images, we will need to somehow break up the image into pieces to put into the separate files.

We want to ensure that the pieces will all be scaled the same, so we'll start off by having `rendersizes` do only half of its work up front—the initial rescaling of the image to the mapper's scale at the default zoom level. We do this simply by using the `prescale` keyword instead of `scale`. Everything else is the same. Since we need to take some areas from the player map and some (like secret doors and passages) from the GM map, we'll prescale both of them equally.

```
$ gma rendersizes prescale 472 16 b2_player_map.png\
 b2_gm_map.png
Converting images to prescale-size image tiles into tiles.map
...
b2_player_map.png: 50 [169.491525423729%]
b2_gm_map.png: 50 [169.491525423729%]
```

For every input file, this produces a pre-scaled image that we can edit further. For our input files of `b2_player_map.png` and `b2_gm_map.png` it outputs `pre_b2_player_map.png` and `pre_b2_gm_map.png`.

We can now take these files and cut out the pieces for each room into separate image files, as shown in Figure 5.8. We used names that were representative of the areas, such as `b2_58.png`, `b2_59g.png`, etc., following the module's room numbers for those areas. Note that although the images are rectangles, we cut out transparent areas where the image will fit around other images if they don't meet at right angles.

Now that we have all the pieces, we let `rendersizes` finish its job by creating the different zoom-level-scaled versions of the images (and preparing them for upload to a server if the `-w` option is given):

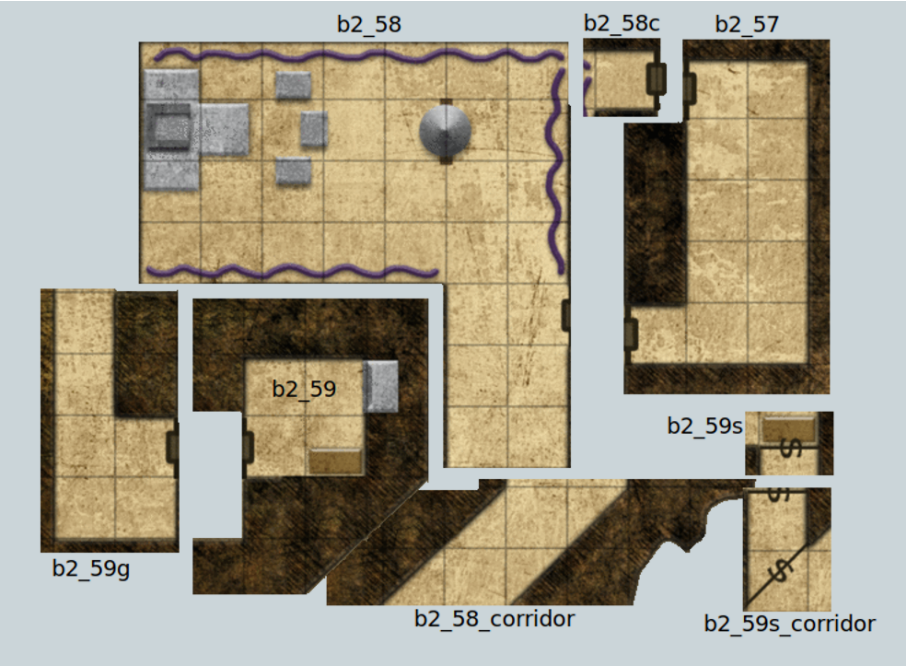


Figure 5.8: Splitting the Map Image Into Parts

```
$ gma rendersizes -L -M B2 -f tiles.map postscale b2_5*
Converting images to postscale-size image tiles into tiles.ma
p...
b2_57.png: 12 [24%] 25 [50%] 50 [100%] 100 [200%] 200 [400%]
b2_58.png: 12 [24%] 25 [50%] 50 [100%] 100 [200%] 200 [400%]
b2_58_corridor.png: 12 [24%] 25 [50%] 50 [100%] 100 [200%] 20
0 [400%]
b2_58c.png: 12 [24%] 25 [50%] 50 [100%] 100 [200%] 200 [400%]
b2_59.png: 12 [24%] 25 [50%] 50 [100%] 100 [200%] 200 [400%]
b2_59g.png: 12 [24%] 25 [50%] 50 [100%] 100 [200%] 200 [400%]
b2_59s.png: 12 [24%] 25 [50%] 50 [100%] 100 [200%] 200 [400%]
b2_59s_corridor.png: 12 [24%] 25 [50%] 50 [100%] 100 [200%] 2
00 [400%]
```

Now when we start up the mapper and load `tiles.map` into it, the mapper knows what files the images are to be found it (or where to fetch them from the server), and we can use them to create the various map files that show each room. This process is essentially the same as what we did before, except we'll be fitting together the image pieces rather than drawing the graphics ourselves (or perhaps a combaination of the two, if we were

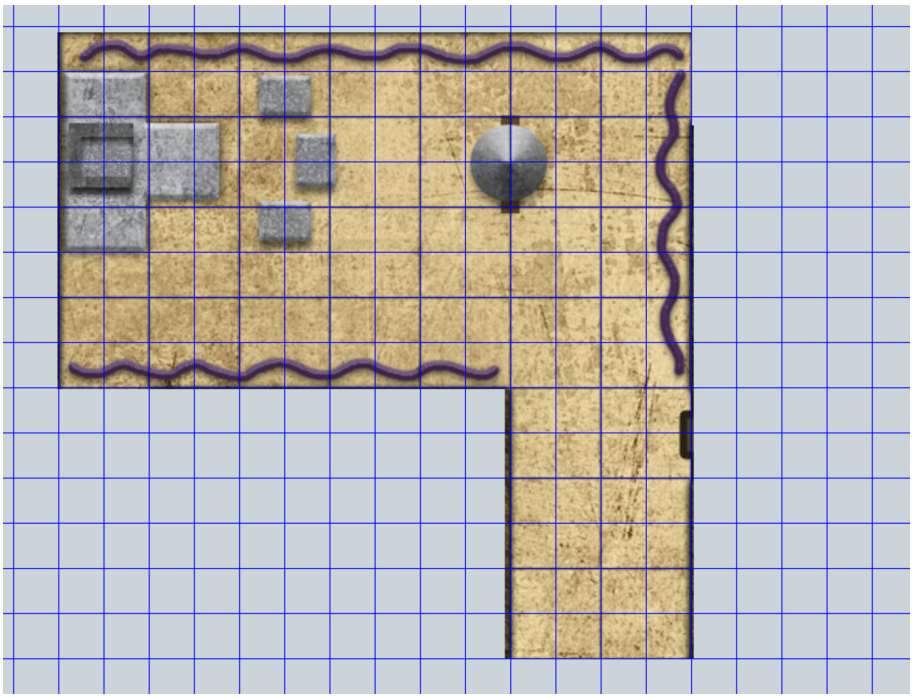


Figure 5.9: Creation of Room `B2_58.map`

drawing some additional elements on top of the graphic tiles).

We'll start with image `b2_58`, which we'll stamp down in the upper-left corner of our map canvas and align with the grid as previously shown. Once we have that, we save the map to a file called `B2_58.map`. (See Figure 5.9.)

Next, we need to find an image that fits against this one. In this case, there is a concealed corridor in the northeast corner leading to room 57, behind the tapestry, so we left it out of the initial map room. Once the party looks behind the tapestry, though, we can reveal it. We'll stamp down image `b2_58c` there now and adjust it until it fits perfectly into place (Figure 5.10).

We don't want to save this as-is, though, since we already have the main room image in its own map file. Additionally, if we came to this little hidden corridor from the other direction, we want to just see it and not the room beyond yet. So before we save this, we need to remove the other room. The easiest way to do this is to unload `B2_58.map` from the mapper by clicking the unload button ( ) and choose the file `B2_58.map`. Now we just have the single little corridor on the map, but it's in just the right place compared to room 58. We save this now into a file called `B2_58c.map`. (See Figure 5.11.)

Room 57 is a little trickier because it needs to connect to both room

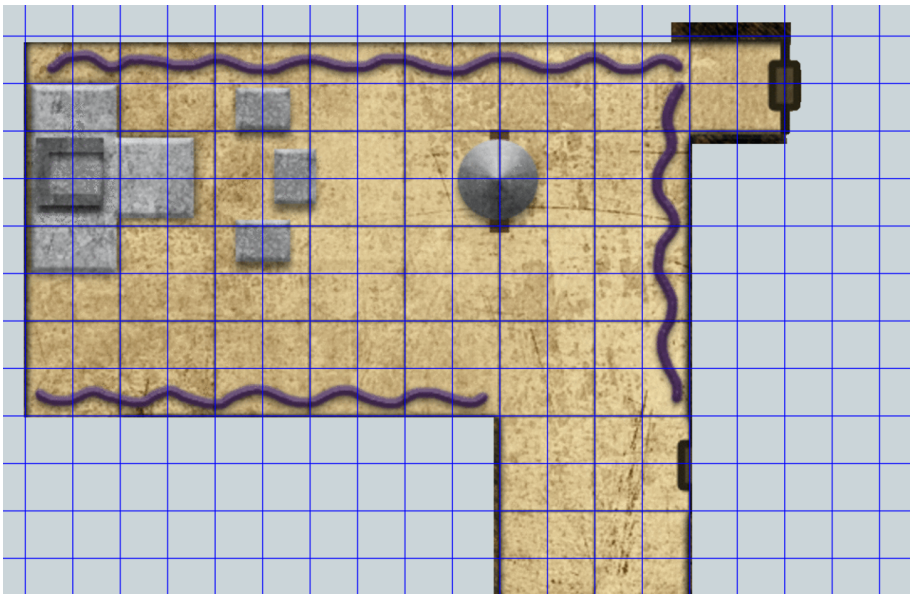


Figure 5.10: Adding Concealed Part of Room `B2_58.map`

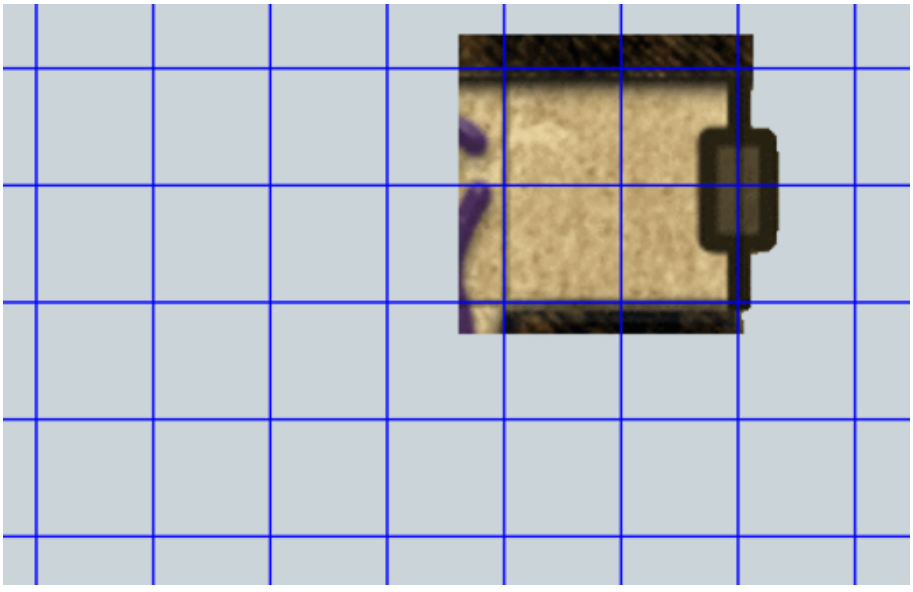


Figure 5.11: Concealed Corridor `B2_58c.map`

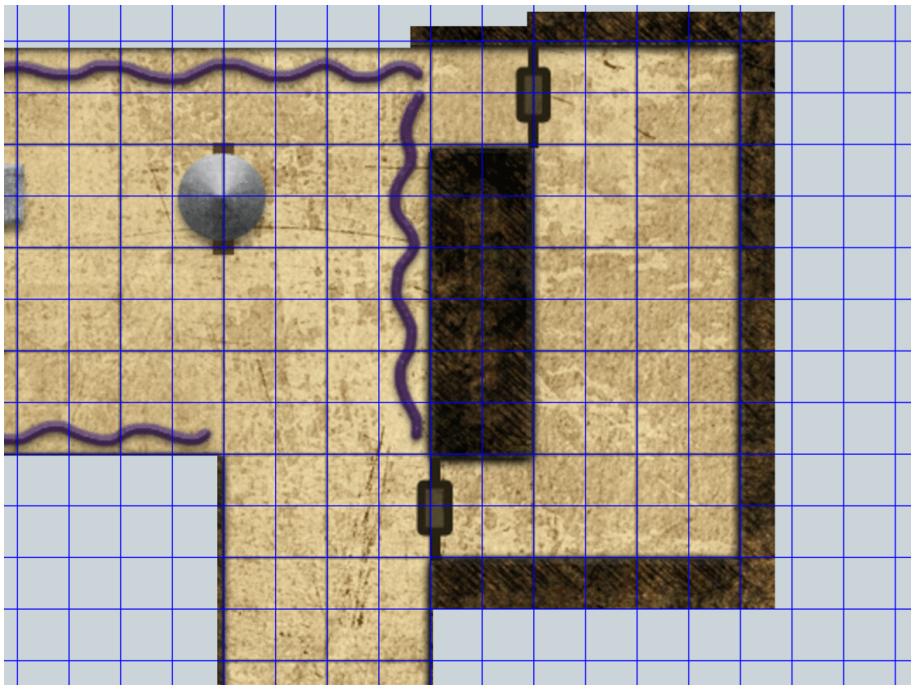


Figure 5.12: Room `B2_57c.map` Connected to Room 58.

58 and the little concealed corridor, so we'll want to have both of those on-screen to verify alignment. Click the merge button ( ) to merge `B2_58.map` back along with the little corridor that's already here. Then stamp down the tile image `b2_57` and align it to connect where it belongs (Figure 5.12).

Once it's in place, we unload `B2_58.map` and `B2_58c.map`, then save the single room into a new file `B2_57.map`.

We repeat the process for the remaining rooms. When we get to room 59, there is a secret door but the image we save to `B2_59.map` is from the player version of the map and does not show anything out of the ordinary. It also includes a solid background so as not to give away the fact that there's something there (tsk, tsk, metagamers...). (See Figure 5.13.)

With room `B2_59.map` loaded, stamp down the image `b2_59s` (the secret door) over the image for room 59's southeast corner. When the secret door is discovered, we'll display that on top of that part of the room to show the now-less-secret door there. (See Figure 5.14.) Unload `B2_59.map` and save that little fragment to `B2_59s.map`.

Finally, we install the diagonal corridor `B2_58_corridor.map` leading into this whole area and the secret passage connecting it to room 59 as `B2_59s_corridor.map`.

During the game, we will merge each of these pieces in whatever order

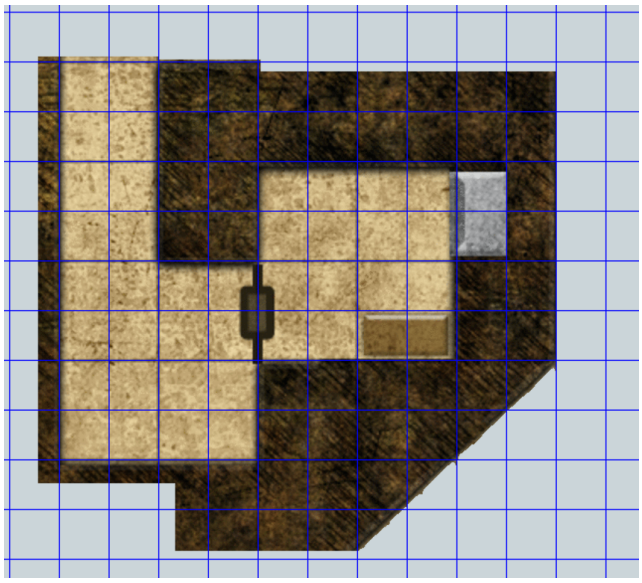


Figure 5.13: Room `B2_57c.map` Rooms 59g and 59, Sans Secret Door.

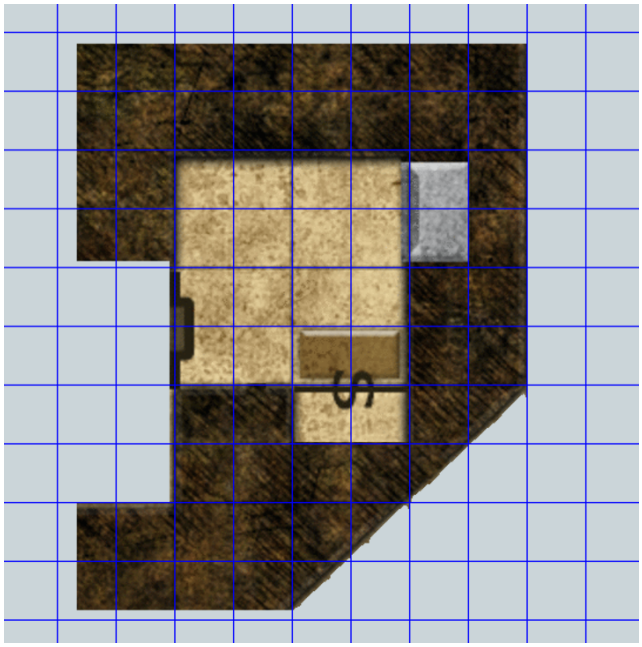


Figure 5.14: Secret Door From Room 59.

the party encounters them. Perhaps they come down the wide corridor into room 58, peek behind the tapestry, then go southwest to 59g, then into room 59. Not finding the secret door, they backtrack and enter room 57. Finally they go back to 59, find the secret door and passage back to the place they started. This sequence is shown in Figure 5.15.

One important thing to note here is the secret passage from room 59 to the diagonal corridor. Once the party discovers that passage, we will display the tiles for the passage over the tiles that showed a plain background. But this may not work out if, while editing the various map files, the secret tile ended up with a lower *z* coordinate (vertical stacking order on the canvas) than the plain background tile did.

The best way to ensure that image will always be on top of everything in its neighborhood is to wait until all of the map files are finished, and then use the `map-translate` program to force that one tile image to the top relative to all the other tiles around it.

The secret tile's image ID is `b2_59s_corridor`, and the secret door itself is `b2_59s`. We'll want the corridor to be on the very top, and the secret door just below it. The only room tiles we really need to consider in this operation are ones which might overlap with those tiles. We will run the program twice, to move the door to the top and then the corridor to the top after it.

```
$ gma map-translate -I b2_59s -z front B2_*.map
$ gma map-translate -I b2_59s_corridor -z front B2_*.map
```

Using Map Images From High-Resolution and/or PDF Files

If the tile images for your map are in a PDF file, `rendersizes` can convert that document to individual graphic image files (one per page) that you can then work with as described above. This makes it easy to work with a high resolution copy of the image, particularly if the original PDF uses scaled graphics so the map may be rendered at arbitrary sizes without pixelation.

To accomplish this, use the `pdf` mode for `rendersizes`, giving the desired dots-per-inch resolution via the `-D` option:

```
$ gma rendersizes -T -D 300 pdf dungeonmaps.pdf
Converting dungeonmaps.pdf from PDF to dungeonmaps.png with
density 300...
```

This will produce 300-dpi image files `dungeonmaps-0.png`, `dungeonmaps-1.png`, etc. The `-T` option trims the edges of each page image.



Figure 5.15: Exploring the Dungeon.

When working with high-resolution images (whether from PDF or not), and using the `prescale` mode of `rendersizes`, best results may be achieved by using the `-m` option:

```
$ gma rendersizes -m prescale 472 16 b2_player_map.png
```

Normally, `rendersizes` will prescale the image to the mapper's default zoom level, so you can work with images at that size. Then, when you run `rendersizes` with the `postscale` mode to deliver the final set of images, it scales them up and down from there to the other zoom factors.

With the `-m` option, however, we tell `rendersizes` to prescale to the maximum resolution. If we do this, we *must* remember to also give the `-m` option to the `postscale` operation:

```
$ gma rendersizes -Lm -M B2 -f tiles.map postscale b2_5*
```

This way, we will work with the highest resolution images, which will all be scaled *down* as needed by the `postscale` operation. This can prevent some loss of image quality if your original images are already at least the highest resolution used by the mapper.

CHAPTER 6

Character Record Sheets

> I keep a file readily accessible with all of my characters' vital stats and what I've actually mentioned in each novel, so that I can maintain consistency throughout the series. Nothing pulls a reader out of the story faster than realizing the author doesn't actually know her own characters.
>
> —Jen Blood

The very first thing a player does in preparation for playing a game such as D&D or Pathfinder is to create a character to play. To keep track of the details of their character, they usually fill out some sort of character record sheet.

The GMA tool suite includes facilities for keeping track of our character records, both for the players' convenience and for the GM to use to maintain a veiw of the party's capabilities and status. As the GMA tool continues to evolve, we plan to have it incorporate more and more the information from the character sheets instead of manual data entry.

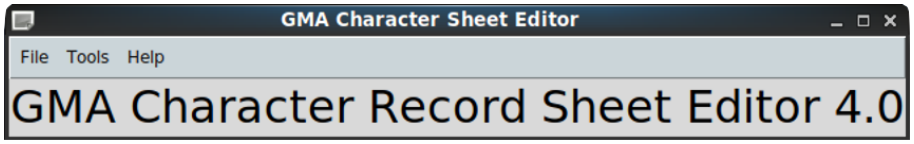
A character record is stored in a GMA Character Record (or `.gcr`) file. This is designed to be easy to edit by hand if necessary, although a tool is under development to provide a better interface.

Editing a Character With Charsheet

The GMA Character Sheet Editor tool (`charsheet`) is invoked with the command

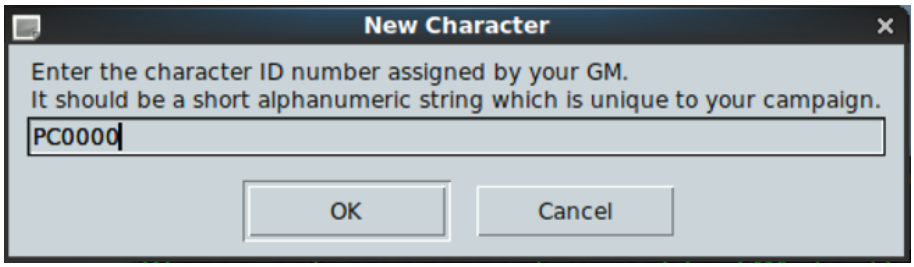
```
$ gma charsheet
```

This brings up the main character sheet editor window. At first (with no character loaded), the main window just has a menu and the tool name.



Once a character is loaded, there will be many tabs full of various statistics and options describing the character.

Select File > New Character to create a new character. This will prompt you to enter the GM-assigned ID for your character.



This is an arbitrary ID but should be unique among the PCs and NPCs in your campaign. This should also be the ID used in the mapper tools to identify your character token.[1]

Since this editor is still under development and not currently in full working order, we won't go into an explanation of its use here yet. For now, character sheets need to be created and edited manually using a text editor.

This is one feature set for GMA where we had grand ambitions for full management of character record data, printing character record sheets, tracking resources in real time during the game, facilitating die-rolls for character actions, and so forth. And then before we got too far along that path, other products came along such as, for example, Lone Wolf Development's Hero Lab software. Since, although GMA is a "labor of love" personal project by the primary developer, it is mostly driven by what features are needed to support his home Pathfinder play group. And since, frankly, most

[1] Don't confuse this with the image name for the token. For example you may have a character with ID `PC42` with image tag `Simon` and token image name `^0^Simon`. It will show as `Simon` on the initiative list and map token label. Internally, the object ID of the token would be `PC42` (or in some protocol commands, `@Simon` will also work).

of his players are using Hero Lab, the development of this part of GMA has stalled for the most part.

Despite this, basic character support as described below *is* actively being developed and maintained to that extent, so it's decent enough for generating paper character record sheets, but lacking in the real-time interactive aspects as well as missing features such as automatic application of the effects of some feats and inventory items on other character stats.

Editing The .gcr Data File Directly

File Format

The .gcr file has a format similar to that of a classic "INI" file. It is organized into *stanzas* which themselves contain *fields*, each of which defines a single attribute of the character. This is very similar to the format we've seen in other chapters for the GMA tool configuration files.

The full details are documented in Appendix C, starting on page 376. Refer to that section for specific guidance as you create and edit the .gcr file. We'll get you started here with the basics.

Creating a New Character

Your GM will assign a unique ID for your character. The author uses numbers that count up sequentially from PC1 but these can be arbitrary and at the GM's discretion. Let's say the GM gives us the ID of EX001.

You will also need a UUID (universally-unique identifier) for your character. You can generate this using a number of standard tools. For example, many systems have a uuidgen utility which hands out new UUIDs for you:

```
$ uuidgen
35c2c859-dda0-4454-abf7-a76a1eb16303
```

If you don't have anything else, you can just use Python, like so:

```
$ python3
Python 3.8.2 (v3.8.2:7b3ab5921f, Feb 24 2020, 17:52:18)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more in
formation.
>>> import uuid
>>> uuid.uuid4()
UUID('bd843373-4b63-4e43-8bc6-c636d2aa5592')
>>>
```

"bd843373-4b63-4e43-8bc6-c636d2aa5592" is the new UUID we just generated.

Now let's start a new file by creating `ex001.gcr`. Put the following in the file to start with:

```
[admin]
GameType=Pathfinder
CharacterUUID=bd843373-4b63-4e43-8bc6-c636d2aa5592
```

This tells GMA that the game system in use is Pathfinder (currently the only supported one), which determines how the rest of the file will be interpreted. We also have our character's UUID listed here.

Assuming this character is Bob the Wizard, let's fill in the bare minimum possible set of values in the file to start with. The `[title]` stanza gives the basic set of stats about Bob, including alignment, class, level, race, and the amount of cash (in units of copper pieces, to keep the value a simple integer) and experience points earned so far.

```
[title]
id=EX001
name=Bob the Wizard
player=Steve
classes=W
levels=1
align=CG
age=20
race=H
ht=5'11"
wt=190#
sex=M
xp=0
cp=0
```

The `[health]` stanza gives information about total hit points, wounds, and the like. Just to get our minimal character sheet, we'll give him a single hit point.

```
[health]
hp=1
```

Saving throws are entered in the `[save]` stanza, but these are mostly calculated for you by GMA. The only thing we need at this point is an empty

`mods` field. This is where we could put notes about special modifiers to saving throws.

```
[save]
mods:
```

Ability scores each have their own stanza. In each case we need, at a minimum, to have a `base` field which gives the base score (as the character was rolled, not including the effects of any magic items or spells that temporarily change these).

```
[str]
base=10
[dex]
base=10
[con]
base=13
[int]
base=17
[wis]
base=16
[cha]
base=11
```

If we print a PostScript sheet from this file, by running:

```
$ gma printchar ex001.gcr
$ ps2pdf ex001.ps ex001.pdf
```

This gives us a PDF file which contains the unimpressive character we see in Figures 6.1 and 6.2.

If the red color scheme doesn't suit you, there are a few others you can choose from by adding a `--color-scheme` option:

```
$ gma printchar --color-scheme blue ex001.gcr
$ ps2pdf ex001.ps ex001.pdf
```

You can specify `black`, `blue`, `green`, `orange`, `red`, or `violet` to get sheets like those in Figure 6.3.

Glancing at the sheet it's obvious we need to fill in more details, but getting to this point first is useful because we can print a new PostScript version at any time as we go to see the effects of our editing if that is helpful.

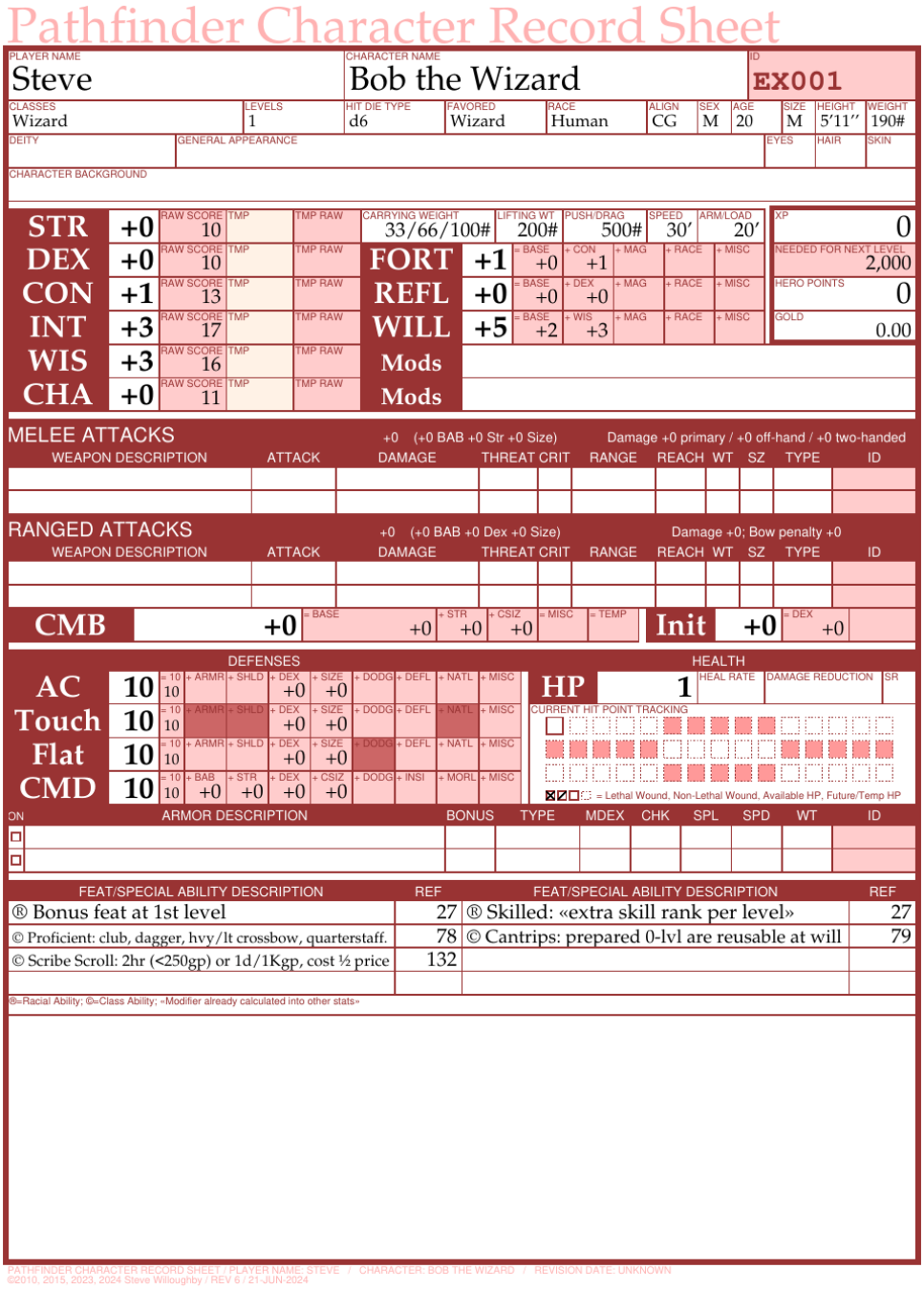


Figure 6.1: Bare Minimum Character Sheet (page 1)

Pathfinder Character Record Sheet

SKILLS

TOTAL POINTS	POINTS/LEVEL	= BASE (CLASS)	+ INT	+ RACE	MAX RANKS (= HD)	ADDED @ LEVEL
6	6	2	+3	+1	1	0

C	T	A	SKILL NAME	MOD	=ABIL	+RANK	+CLS	+SYN	+RACE	+ITEM	+MISC	SKILL DESCRIPTION
☐	☐	☒	Acrobatics	+0	DX +0							balance, jump, roll, etc.
☒	☐	☐	Appraise	+3	IN +3							evaluate monetary value of objects.
☐	☐	☐	Bluff	+0	CH +0							deceive opponent (vs. their Sense Motive)
☐	☐	☒	Climb	+0	ST +0							scale vertical surfaces.
☐	☐	☐	Diplomacy	+0	CH +0							persuade others to agreement.
☐	☐	☐	Disguise	+0	CH +0							change your appearance.
☐	☐	☒	Escape Artist	+0	DX +0							slip through squeeze/bond/grapple.
☒	☐	☒	Fly	+0	DX +0							maneuver/combat skill in flight.
☐	☐	☐	Heal	+3	WI +3							stabilize dying person/first aid.
☐	☐	☐	Intimidate	+0	CH +0							frighten others to comply with you.
☐	☐	☐	Perception	+3	WI +3							notice things, search, &c.
☐	☐	☒	Ride	+0	DX +0							effectively ride a mount.
☐	☐	☐	Sense Motive	+3	WI +3							detect falsehoods & true intent.
☐	☐	☒	Stealth	+0	DX +0							hide, move silently, sniping.
☐	☐	☐	Survival	+3	WI +3							survive in wild, track, navigate.
☐	☐	☒	Swim	+0	ST +0							½ spd (or ¼) each round
☐	☐	☐	*** ALLOCATE RANKS ***									*** 6 MORE TO USE ***

C=Class skill; T=Training Required; A=Armor Check Penalty (Swim=2)

POSSESSIONS

TOTAL WEIGHT 0#

DESCRIPTION	LOC	WT	ID

PATHFINDER CHARACTER RECORD SHEET / PLAYER NAME: STEVE / CHARACTER: BOB THE WIZARD / REVISION DATE: UNKNOWN

©2010, 2015, 2023, 2024 Steve Willoughby / REV 6 / 21-JUN-2024

Figure 6.2: Bare Minimum Character Sheet (page 2)



Figure 6.3: Character Sheet Color Schemes

We also notice that the skill section of the printed form is telling us we need to assign more ranks. This is one of the very few cases where messages about the validity of the file itself are printed on the sheet.

We can add a `notes` field to the `[title]` stanza to write some free-form notes about the character that don't necessarily fit elsewhere:

```
[title]
notes:
    Bob is looking for his lost cat.
    |
    He's also looking for a new //wand of magic missiles//
```

This is an example of a *multi-line field*. All lines after the first are indented to show that they are continuing the field onto multiple lines rather than starting a new one.

Since the `notes` field is typeset as paragraphs of running text, we put a blank line in the middle of the notes to start a new paragraph. However, we can't have a blank line in the `.gcr` file in the middle of a field (because that's not allowed by the INI file format we're using), so we represent that by having (an indented) line with just a vertical bar. The result looks like this:

CHARACTER NOTES

Bob is looking for his lost cat.

He's also looking for a new *wand of magic missiles*

If we had left that out, like this

```
[title]
notes:
    Bob is looking for his lost cat.
    He's also looking for a new //wand of magic missiles//
```

then the lines would flow together in the same paragraph, with this result:

CHARACTER NOTES

Bob is looking for his lost cat. He's also looking for a new *wand of magic missiles*

Let's finish up the `[title]` stanza a bit now, making Bob a 6th level wizard.

```
[title]
id=EX001
name=Bob the Wizard
nickname=Bob
```

```
player=Steve
classes=W
levels=6
align=CG
race=H
sex=M
age=26
ht=5'11"
wt=190#
eyes=Blue
hair=Black
skin=Fair
deity=Nethis
appearance=Unassuming, simply dressed in grey robes
background=Quiet, non-aggressive, scholarly type;
    exploring frontiers of new magics. See attached for more.
cp=54020
xp=23000
notes:
    Bob is looking for his lost cat.
    |
    He's also looking for a new //wand of magic missiles//
```

Doing so produces this title block on the character sheet:

PLAYER NAME		CHARACTER NAME				ID				
Steve		Bob the Wizard				EX001				
CLASSES	LEVELS	HIT DIE TYPE	FAVORED	RACE	ALIGN	SEX	AGE	SIZE	HEIGHT	WEIGHT
Wizard	6	d6	Wizard	Human	CG	M	26	M	5'11"	190#
DEITY	GENERAL APPEARANCE						EYES	HAIR	SKIN	
Nethis	Unassuming, simply dressed in grey robes						Blue	Black	Fair	
CHARACTER BACKGROUND										
Quiet, non-aggressive, scholarly type; exploring frontiers of new magics. See attached for more.										

Classes are shown using defined codes for each, like `F` for Fighter or `W` for Wizard. If the character is multi-classed, such as a Fighter/Wizard, list them separated by slashes. The level field shows the level in the same manner. So a 4th level Wizard/3rd level Fighter would be listed:

```
[title]
classes=W/F
levels=4/3
```

We already have Bob's ability scores and saving throws entered, so we'll leave those alone. The calculations made by GMA produced these values from the information entered:

STR	+0	RAW SCORE 10	TMP	TMP RAW	CARRYING WEIGHT 33/66/100#	LIFTING WT 200#	PUSH/DRAG 500#	SPEED 30′	ARM/LOAD 20′	XP 23,000								
DEX	+0	RAW SCORE 10	TMP	TMP RAW	FORT	+3	= BASE +2	+ CON +1	+ MAG	+ RACE	+ MISC	NEEDED FOR NEXT LEVEL 35,000						
CON	+1	RAW SCORE 13	TMP	TMP RAW	REFL	+2	= BASE +2	+ DEX +0	+ MAG	+ RACE	+ MISC	HERO POINTS 0						
INT	+3	RAW SCORE 17	TMP	TMP RAW	WILL	+8	= BASE +5	+ WIS +3	+ MAG	+ RACE	+ MISC	GOLD 540.20						
WIS	+3	RAW SCORE 16	TMP	TMP RAW	Mods													
CHA	+0	RAW SCORE 11	TMP	TMP RAW	Mods													

Combat Block

The middle section of the first page deals with all your character's combat capabilities. The first section is for making attacks.

Most of this section is automatically computed for you. You do, however, need to specify your armor and weapons. List the weapons currently in active use in the `[weapons]` stanza's active field:

```
[weapons]
active=123 124
```

Then, for each weapon owned by the character, create a stanza called `[weapon.`⟨*id*⟩`]`, which includes at least the type of weapon, and possibly a `sz` field indicating size code (if not medium-size). Additional fields may be given if the specific instance of weapon is different from standard-issue weapons of its type. Note here that Bob possesses a scythe (weapon ID #125), but isn't carrying it around with him, so it does not appear in the `active` list or on the character sheet, but is recorded in his character data file:

```
[weapon.123]
type=SwordShort
[weapon.124]
type=Starknife
[weapon.125]
type=Scythe
```

This gives us the following attack block:

MELEE ATTACKS		+3 (+3 BAB +0 Str +0 Size)			Damage +0 primary / +0 off-hand / +0 two-handed					
WEAPON DESCRIPTION	ATTACK	DAMAGE	THREAT	CRIT	RANGE	REACH	WT	SZ	TYPE	ID
Sword, short	+3	1d6	19–20	×2			2#	M	P	123
Starknife	+3	1d4	20	×3			3#	M	P	124

RANGED ATTACKS		+3 (+3 BAB +0 Dex +0 Size)			Damage +0; Bow penalty +0					
WEAPON DESCRIPTION	ATTACK	DAMAGE	THREAT	CRIT	RANGE	REACH	WT	SZ	TYPE	ID
Starknife	+3	1d4	20	×3	5× 20′		3#	M	P	124

The block is divided into melee and ranged attacks. Since the starknife can be used as both a melee weapon (to stab with) and a ranged weapon (by throwing it at someone), it appears in both blocks.

See the full documentation for the `.gcr` file format starting on page 376 for more information about entering custom weapons that aren't already known to GMA, modifying whether a weapon counts as ranged or melee.

Bob is a universalist wizard, so he has a class ability called Hand of the Apprentice which lets him throw weapons using magic, using his Intelligence modifier for the attack. To represent this, we need a third kind of attack, since this is calculated differently from melee and ranged attacks.

To do this, add a `modes` field to the `[weapons]` stanza along with a corresponding new section which defines the new mode in the `.gcr` file:

```
[weapons]
active=123 124
modes=hand

[mode.hand]
name=Hand of the Apprentice
ability=int
active=123 124
is_melee=true
```

This creates the new attack mode and indicates that weapons 123 and 124 may be used with it. We also indicate that this counts as a melee attack for the purposes of calculating damage since that's how Hand of the Apprentice is described in the rules.

This adds the following additional attack block to the character sheet:

Hand of the Apprentice		+6 (+3 BAB +3 Int +0 Size)			Damage +0 primary / +0 off-hand / +0 two-handed					
WEAPON DESCRIPTION	ATTACK	DAMAGE	THREAT	CRIT	RANGE	REACH	WT	SZ	TYPE	ID
Sword, short	+6	1d6	19–20	×2			2#	M	P	123
Starknife	+6	1d4	20	×3	5× 20′		3#	M	P	124

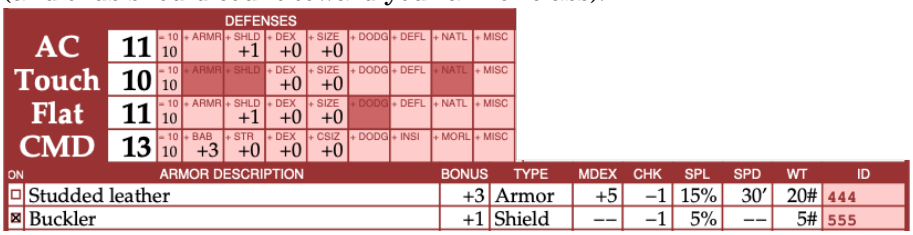
The end of the attack block includes the character's combat maneuver bonus and initiative modifier.

CMB	+3	= BASE	+ STR	+ CSIZ	= MISC	= TEMP	Init	+0	= DEX	
		+3	+0	+0					+0	

Armor is handled similarly.

```
[armour]
active=444 555
[armour.444]
type=StuddedLeather
[armour.555]
type=Buckler
on=yes
```

Note the `on` field which indicates that the armor is currently being worn (and thus should count toward your armor class).



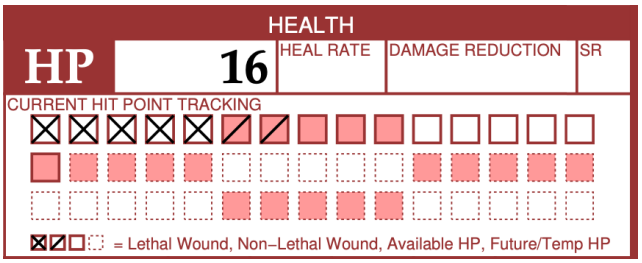
DEFENSES												
AC	11	= 10 10	+ ARMR	+ SHLD +1	+ DEX +0	+ SIZE +0	+ DODG	+ DEFL	+ NATL	+ MISC		
Touch	10	= 10 10	+ ARMR	+ SHLD	+ DEX +0	+ SIZE +0	+ DODG	+ DEFL	+ NATL	+ MISC		
Flat	11	= 10 10	+ ARMR	+ SHLD +1	+ DEX +0	+ SIZE +0	+ DODG	+ DEFL	+ NATL	+ MISC		
CMD	13	= 10 10	+ BAB +3	+ STR +0	+ DEX +0	+ CSIZ +0	+ DODG	+ INSI	+ MORL	+ MISC		

ON	ARMOR DESCRIPTION	BONUS	TYPE	MDEX	CHK	SPL	SPD	WT	ID
☐	Studded leather	+3	Armor	+5	−1	15%	30′	20#	444
☒	Buckler	+1	Shield	−−	−1	5%	−−	5#	555

Health Block

We roll up our hit points and enter them in the `[health]` stanza. If there are wounds, we note that too.

```
[health]
hp=16
wounds=5
nonlethal=2
```



Special Abilities and Feats

The special abilities are in a stanza called `[special]`. If there are any racial traits or class abilities which require you to specify a choice about something, there will be a field for that in this stanza. For example, If you are a sorcerer with the draconic bloodline, you'd have this:

```
[special]
Bloodline=draconic
```

Otherwise, the feats your character may have are listed in the feats field, like this:

```
[special]
feats:
    ImprovedInitiative
    SpellPenetration
    Languages|Common, Draconic, Elven
```

Feats or special abilities with parameters are separated from them with vertical bars. Let's fill in Bob's list of feats now.

```
[special]
feats:
    Languages|Common, Elven, Celestial, Draconic, Orc, Sylvan
    ArcaneBond|familiar|owl
    ArcaneSchool|Universalist
    HandOfTheApprentice
    CombatCasting
    EschewMaterials
    ImprovedInitiative
    HeightenSpell
```

which prints like this:

FEAT/SPECIAL ABILITY DESCRIPTION	REF	FEAT/SPECIAL ABILITY DESCRIPTION	REF
Languages: Common, Elven, Celestial, Draconic, Orc, Sylvan		© Arcane Bond (Ex/Sp): familiar: owl	78
© Arcane School: Universalist	78	© Hand of the Apprentice (Su): weap 30' (use INT mod: +3) 6/day	82
Combat Casting: +4 Concentration	119	Eschew Materials: M component <=1 gp not needed	123
Improved Initiative: «+4 to initiative»	127	Heighten Spell: Prepare/cast spell as if higher level	126
® Bonus feat at 1st level	27	® Skilled: «extra skill rank per level»	27
© Proficient: club, dagger, hvy/lt crossbow, quarterstaff.	78	© Cantrips: prepared 0-lvl are reusable at will	79
© Scribe Scroll: 2hr (<250gp) or 1d/1Kgp, cost ½ price	132		
®=Racial Ability; ©=Class Ability; «Modifier already calculated into other stats»			

Note that several racial traits (marked with "®") and class abilities (with "©") were added automatically by the system. The others were manually entered.

Note the effects enclosed in angle quotes (e.g., "Improved Initiative: «+4 to initiative»"). This means that the system automatically applied that effect to the character sheet, and you don't need to add it again (unless special circumstances require it). Having added that feat, Bob's initiative block now looks like this:

Init	+4	= DEX +0	+ MISC +4

Skills Sheet

The skills are listed in the `[skills]` stanza. Bonus points due to levelling in a favored class are entered like this:

```
[skills]
bonus=6
```

Each skill in which your character is trained goes in a stanza of its own. It is named `[skill.`⟨*name*⟩`]` (and skills with sub-categories are in stanzas named `[skill.`⟨*name*⟩`.`⟨*subname*⟩`]`). For example:

```
[skill.fly]
ranks=6
[skill.craft.traps]
ranks=1
```

Bob's skills are entered as:

```
[skills]
bonus=6
[skill.craft.alchemy]
ranks=3
```

```
[skill.knowledge.planes]
ranks=6
[skill.knowledge.arcana]
ranks=6
[skill.perception]
ranks=4
[skill.spellcraft]
ranks=6
[skill.survival]
ranks=2
[skill.diplomacy]
ranks=3
[skill.heal]
ranks=3
[skill.linguistics]
ranks=3
[skill.appraise]
ranks=3
[skill.stealth]
ranks=3
```

The skill list on Bob's character sheet then looks like this:

SKILLS

TOTAL POINTS	POINTS/LEVEL	= BASE (CLASS)	+ INT	+ RACE	MAX RANKS (= HD)	ADDED @ LEVEL
42	6	2	+3	+1	6	6

C	T	A	SKILL NAME	MOD	=ABIL	+RANK	+CLS	+SYN	+RACE	+ITEM	+MISC	SKILL DESCRIPTION
☐	☐	☒	Acrobatics	**+0**	DE+0							balance, jump, roll, etc.
☒	☐	☐	Appraise	**+9**	IN+3	+3	+3					evaluate monetary value of objects.
☐	☐	☐	Bluff	**+0**	CH+0							deceive opponent (vs. their Sense Motive)
☐	☐	☒	Climb	**+0**	ST+0							scale vertical surfaces.
☒	☐	☐	Craft (Alchemy)	**+9**	IN+3	+3	+3					make items or earn (½ chk) gp/wk
☐	☐	☐	Diplomacy	**+3**	CH+0	+3						persuade others to agreement.
☐	☐	☐	Disguise	**+0**	CH+0							change your appearance.
☐	☐	☒	Escape Artist	**+0**	DE+0							slip through squeeze/bond/grapple.
☒	☐	☒	Fly	**+0**	DE+0							maneuver/combat skill in flight.
☐	☐	☐	Heal	**+6**	WI+3	+3						stabilize dying person/first aid.
☐	☐	☐	Intimidate	**+0**	CH+0							frighten others to comply with you.
☒	☒	☐	Knowledge (Arcana)	**+12**	IN+3	+6	+3					ancient mysteries, magic, symbols, &c.
☒	☒	☐	Knowledge (Planes)	**+12**	IN+3	+6	+3					planes, outsiders, planar magic.
☒	☒	☐	Linguistics	**+9**	IN+3	+3	+3					decipher writing, language, forgery, &c.
☐	☐	☐	Perception	**+7**	WI+3	+4						notice things, search, &c.
☐	☐	☒	Ride	**+0**	DE+0							effectively ride a mount.
☐	☐	☐	Sense Motive	**+3**	WI+3							detect falsehoods & true intent.
☒	☒	☐	Spellcraft	**+12**	IN+3	+6	+3					cast/identify spells/objects.
☐	☐	☒	Stealth	**+3**	DE+0	+3						hide, move silently, sniping.
☐	☐	☐	Survival	**+5**	WI+3	+2						survive in wild, track, navigate.
☐	☐	☒	Swim	**+0**	ST+0							½ spd (or ¼) each round

C=Class skill; T=Training Required; A=Armor Check Penalty (Swim=2)

Inventory of Possessions

This is pretty simple, just a single list of items, one per line, like this:

```
[inventory]
items:
    16| 1818|belt |Longsword        |   |   | ||GenericItem|1
    17|  909|worn |Shortbow         |   |   | ||GenericItem|1
    18| 1818|belt |Mace, light      |   |   | ||GenericItem|1
    19| 4545|worn |Shield, light, wooden|  || ||GenericItem|1
    20| 6818|stored|Leather armor   |   |   | ||GenericItem|0
    21| 6818|quiver|Arrows          |100|100|2||GenericItem|0
    22|  454|pack |Torch            |  1|  1| ||Torch      |1
    23|  909|worn |Backpack         |   |   | ||Backpack   |1
    24| 4545|pack |Rope, 50'        |   |   | ||RopeSilk   |1
    25| 4545|pack |Food, rations    | 10| 10|||RationsTrail|1
    26|  454|worn |Holy symbol      |  ||||HolySymbolWooden|1
```

Each line consists of four or more fields separated by vertical bars. They are, in order:

1. Item ID if any given by the GM.
2. Weight in grams
3. Location of the item
4. Description of the item
5. Number of check-box squares to make room for (these will be dashed outlines on the form)
6. Number of check-box squares to actually show as representing items in the set
7. Number of boxes checked off now
8. Number of boxes half-checked off now
9. Item class or `GenericItem`
10. 1 if the item is being carried or 0 if not.

With the items listed above, Bob's inventory prints like this:

POSSESSIONS TOTAL WEIGHT 44# 1 oz

DESCRIPTION	LOC	WT	ID
Longsword	belt	4#	16
Shortbow	worn	2#	17
Mace, light	belt	4#	18
Shield, light, wooden	worn	10#	19
Leather armor	stored	0#	20
Arrows	quiver	0#	21
Torch	pack	1#	22
Backpack	worn	2#	23
Rope, 50′	pack	10#	24
Food, rations	pack	10#	25
Holy symbol	worn	1#	26

Spells

Basic spell data are entered in the `[spells]` stanza. In it, you list all the types of magic you can cast. Bob casts only wizard spells, so his looks like this:

```
[spells]
types=wizard
```

For every type of spell you can cast, you also add a specific stanza for that type of spell. This includes domains, which add their own spell lists into play. Within each stanza, you can select all spells up to and including a certain level:

```
[spells.cleric]
all_level=3
```

or all spells up to that level except for those of a certain type:

```
[spells.cleric]
all_level=3
except=evil
```

or you can list the specific spells by name for each spell level using the Roman numerals for each level:

```
[spells.wizard]
all_level=0
I:
Alarm
Hold Portal
Summon Monster I
II:
Acid Arrow
Hideous Laughter
```

See the detailed documentation starting on p. 376 for more information about specifying metamagic effects and which spells have been prepared and cast.

Printing a Paper Character Sheet

The `printchar` program will read a `.gcr` file and generate a PostScript version in an output file of the same name but with a `.ps` suffix. This may be printed by a printer or program which understands PostScript or may be converted to another format such as PDF.

```
$ gma printchar ex001.gcr
$ ps2pdf ex001.ps ex001.pdf
```

There are options to control things like the color scheme

```
$ gma printchar --color-scheme blue ex001.gcr
$ ps2pdf ex001.ps ex001.pdf
```

or to have all skills printed (by default, only skills available to the character are printed)

```
$ gma printchar --all-skills blue ex001.gcr
$ ps2pdf ex001.ps ex001.pdf
```

In addition, this program may be used to generate a blank data file which you can use to begin editing a new sheet

```
$ gma printchar --blank-data new_char.gcr
```

If you want a blank paper form which you can print in order to manually write on (as D&D players have been doing since the 1970s), run this:

```
$ gma printchar --blank-ps blank.ps
$ ps2pdf blank.ps blank.pdf
```

You can also generate a hardcopy of a character record sheet with all the most important fields highlighted in yellow. This can help, for example, newer players since it points out the parts of the sheet they need to become familiar with first and will be referring to most often.

```
$ gma printchar --highlight-important-areas ex001.gcr
$ ps2pdf ex001.ps ex001.pdf
```

This produces output like that shown in Figure 6.4.

Party Skill Summaries

For the GM's reference, a summary sheet can be printed with the collected statistics of a party full of characters.

To start, create a party list file (for our example, let's call it "`party.conf`") which lists the character `.gcr` files of the party members:

```
[party]
members=Avia Nolin Rigel Trask
[files]
characters=/opt/gma/party
```

This file indicates that the party's `.gcr` files are stored in the directory `/opt/gma/party`. The characters to include in the summary report are Avia, Nolin, Rigel, and Trask. For example, the file for Avia is located at `/opt/gma/party/Avia.gcr`.

If your character files are stored in more than one directory, just list them on separate lines in the `characters` field, indenting each line after the first.

```
[files]
characters=/opt/gma/party
    /usr/local/games/characters
```

In this case, each character file will be looked for in `/opt/gma/party` and in `/usr/local/games/characters`, in that order, until a file with the suffix `.gcr` or `.por` is found.

Pathfinder Character Record Sheet

PLAYER NAME	CHARACTER NAME	ID
Steve	Bob the Wizard	EX001

CLASSES	LEVELS	HIT DIE TYPE	FAVORED	RACE	ALIGN	SEX	AGE	SIZE	HEIGHT	WEIGHT
Wizard	6	d6	Wizard	Human	CG	M	26	M	5′11″	190#

DEITY	GENERAL APPEARANCE	EYES	HAIR	SKIN
Nethis	Unassuming, simply dressed in grey robes	Blue	Black	Fair

CHARACTER BACKGROUND: Quiet, non-aggressive, scholarly type; exploring frontiers of new magics. See attached for more.

		RAW SCORE	TMP	TMP RAW
STR	+0	10		
DEX	+0	10		
CON	+1	13		
INT	+3	17		
WIS	+3	16		
CHA	+0	11		

CARRYING WEIGHT 33/66/100# | LIFTING WT 200# | PUSH/DRAG 500# | SPEED 30′ | ARM/LOAD 20′

		= BASE	+ CON	+ MAG	+ RACE	+ MISC
FORT	+3	+2	+1			
REFL	+2	+2	+0			
WILL	+8	+5	+3			
Mods						
Mods						

XP 23,000 | NEEDED FOR NEXT LEVEL 35,000 | HERO POINTS 0 | GOLD 540.20

MELEE ATTACKS — +3 (+3 BAB +0 Str +0 Size) — Damage +0 primary / +0 off-hand / +0 two-handed

WEAPON DESCRIPTION	ATTACK	DAMAGE	THREAT	CRIT	RANGE	REACH	WT	SZ	TYPE	ID
Sword, short	+3	1d6	19–20	×2			2#	M	P	123
Starknife	+3	1d4	20	×3			3#	M	P	124

RANGED ATTACKS — +3 (+3 BAB +0 Dex +0 Size) — Damage +0; Bow penalty +0

WEAPON DESCRIPTION	ATTACK	DAMAGE	THREAT	CRIT	RANGE	REACH	WT	SZ	TYPE	ID
Starknife	+3	1d4	20	×3	5× 20′		3#	M	P	124

Hand of the Apprentice — +6 (+3 BAB +3 Int +0 Size) — Damage +0 primary / +0 off-hand / +0 two-handed

WEAPON DESCRIPTION	ATTACK	DAMAGE	THREAT	CRIT	RANGE	REACH	WT	SZ	TYPE	ID
Sword, short	+6	1d6	19–20	×2			2#	M	P	123
Starknife	+6	1d4	20	×3	5× 20′		3#	M	P	124

CMB +3 (= BASE +3 | + STR +0 | + CSIZ +0 | = MISC | = TEMP)

Init +4 (= DEX +0 | + MISC +4)

DEFENSES

		= 10	+ ARMR	+ SHLD	+ DEX	+ SIZE	+ DODG	+ DEFL	+ NATL	+ MISC
AC	11	10		+1	+0	+0				
Touch	10	10			+0	+0				
Flat	11	10		+1	+0	+0				
CMD	13	10	+ BAB +3	+ STR +0	+ DEX +0	+ CSIZ +0	+ DODG	+ INSI	+ MORL	+ MISC

HEALTH: HP 16 | HEAL RATE | DAMAGE REDUCTION | SR

CURRENT HIT POINT TRACKING

☒☒ = Lethal Wound, Non-Lethal Wound, Available HP, Future/Temp HP

ON	ARMOR DESCRIPTION	BONUS	TYPE	MDEX	CHK	SPL	SPD	WT	ID
☐	Studded leather	+3	Armor	+5	–1	15%	30′	20#	444
☒	Buckler	+1	Shield	—	–1	5%	—	5#	555
☐									
☐									

FEAT/SPECIAL ABILITY DESCRIPTION	REF	FEAT/SPECIAL ABILITY DESCRIPTION	REF
Languages: Common, Elven, Celestial, Draconic, Orc, Sylvan		© Arcane Bond (Ex/Sp): familiar: owl	78
© Arcane School: Universalist	78	© Hand of the Apprentice (Su): weap 30′ (use INT mod: +3) 6/day	82
Combat Casting: +4 Concentration	119	Eschew Materials: M component <=1 gp not needed	123
Improved Initiative: «+4 to initiative»	127	Heighten Spell: Prepare/cast spell as if higher level	126
® Bonus feat at 1st level	27	® Skilled: «extra skill rank per level»	27

PATHFINDER CHARACTER RECORD SHEET / PLAYER NAME: STEVE / CHARACTER: BOB THE WIZARD / REVISION DATE: UNKNOWN
©2010, 2015, 2023, 2024 Steve Willoughby / REV 6 / 21-JUN-2024

Figure 6.4: Character Sheet with Highlighted Areas

We can generate a player report which shows what the characters would have come to know about their associates, in vague, general terms, by running this command:

```
$ gma skillranks --conf=party.conf --playerreport
```

This ranks each character from best to worst by ability scores and their skills. If there is a significant gap, a slash is placed between characters. So in the output in Figure 6.5, Avia is the strongest, followed by Nolin, Trask, and Rigel but there is a bit of a gap between Nolin's and Trask's strength while Avia and Nolin are close to each other in strength, as are Trask and Rigel.

For the GM's reference, a more detailed PostScript matrix report can be generated which summarizes character stats including abilities, saving throws, armor class, hit points, and skill modifiers.

```
$ gma skillranks --conf=party.conf --matrix=party.ps
$ ps2pdf party.ps party.pdf
```

This produces the output shown in Figure 6.6. You can also add free-form notes that will appear under the skills matrix as documented on page 557.

Importing Character Data from Hero Lab Portfolio Files

The `skillranks` GMA command can also read character stats from Hero Lab portfolio (`.por`) files just as it does with GMA character record (`.gcr`) files. This allows you to give your players the flexibility to use either tool to manage their character files while still allowing you to pull the data you need to run encounters and initiative in GMA.

N.B.: Note that the support for reading from `.por` files is still experimental.

The data are read from the "statblock" data contained in the `.por` file. Hero Lab can record this data in XML, HTML, and/or plain text forms. The preferred data format is XML. If that is not found, `skillranks` will do its best to interpret the data from the plain text statblock instead.

Importing Character Data Into GMA

As we saw in on page 17, we need to set up some basic information about our party's characters into the GMA preferences editor to pre-seed the initiative list for each encounter and to provide spell caster level information for setting up the duration timers for spells.

We can automate this a bit by importing the character record data directly from the `.gcr` files (and/or `[char.`⟨*id*⟩`]` stanzas of the `skillranks`

```
$ gma skillranks --conf=party.conf --playerreport
*** Party Member Relative Abilities ***
                     Strongest: Avia, Nolin / Trask, Rigel
                      Nimblest: Rigel / Avia, Trask / Nolin
                    Healthiest: Trask, Avia, Rigel, Nolin
                      Smartest: Avia, Trask, Rigel, Nolin
                        Wisest: Nolin, Avia, Rigel, Trask
              Most Charismatic: Trask / Avia / Nolin, Rigel
                    Acrobatics: Rigel / Trask / Avia / Nolin
                      Appraise: Rigel / Trask / Avia, Nolin
                         Bluff: Trask / Rigel / Avia, Nolin
                         Climb: Nolin, Rigel / Avia / Trask
                 Craft (Armor): Avia
                Craft (Jewelry): Nolin
           Craft (Stonemasonry): Avia
                Craft (Weapons): Avia
                     Diplomacy: Trask, Avia / Nolin, Rigel
                Disable Device: Rigel / Avia
                      Disguise: Trask / Avia, Nolin, Rigel
                Escape Artist: Rigel / Trask / Avia / Nolin
                          Fly: Avia, Trask / Rigel / Nolin
                Handle Animal: Nolin / Avia
                         Heal: Avia / Trask, Nolin, Rigel
                   Intimidate: Trask / Avia, Nolin / Rigel
            Knowledge (Arcana): Trask / Avia
      Knowledge (Dungeoneering): Rigel / Nolin
        Knowledge (Engineering): Nolin / Avia
          Knowledge (Geography): Nolin
            Knowledge (History): Avia
              Knowledge (Local): Trask
            Knowledge (Nature): Nolin
          Knowledge (Nobility): Avia
          Knowledge (Religion): Avia
                   Linguistics: Avia, Rigel, Nolin
                    Perception: Avia, Rigel / Trask / Nolin
                          Ride: Nolin / Avia, Trask, Rigel
                  Sense Motive: Rigel / Avia / Nolin / Trask
               Sleight Of Hand: Rigel / Avia
                    Spellcraft: Trask / Avia
                       Stealth: Rigel / Trask, Avia / Nolin
                      Survival: Nolin / Avia, Trask / Rigel
                          Swim: Nolin, Avia, Rigel / Trask
              Use Magic Device: Trask / Rigel, Avia
```

Figure 6.5: Player Skill Rank List

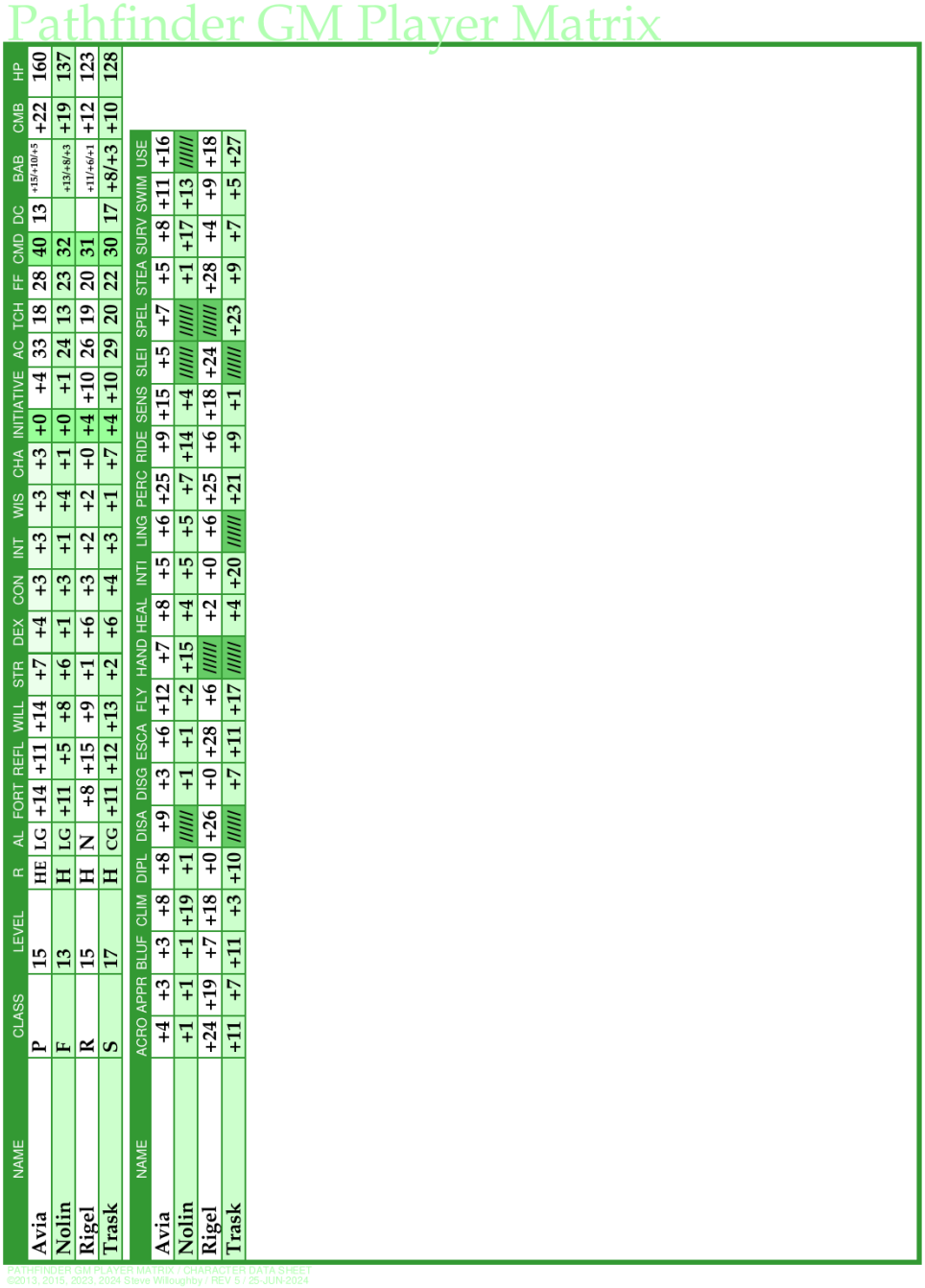


Figure 6.6: GM Skill Rank Matrix

configuration file) into the GMA preferences using the `gma skillranks` command.

After putting together the party configuration as described above, run `gma skillranks` with the `--world` option. The parameter to this option is the name of the existing campaign world into which you want to update character information.

If you also include the `--prune` option, all the existing party information will be removed first. Otherwise, any characters already in the indicated world which are not mentioned in the `skillranks` configuration will remain as-is.

If your preferences file isn't in the default location, the `--preferences` option can change that.

```
$ gma skillranks --conf=party.conf --world=example
```

CHAPTER 7

Roll Initiative: Preparing Encounters in Advance

> The monster rolls a...
> nevermind, your turn.
>
> —Every GM

In chapter 3 we looked at running combat using the GM console, manually adding monsters and PCs to the initiative list, rolling HP for each, and so forth. However, with a little pre-game preparation, the GM can take advantage of some additional capabilities of GMA to save time at the game table by avoiding the overhead of manual data entry.

Basic Encounters and the `.gmo` File

In Chapter 6 we saw how a PC or NPC could be represented by a character sheet data file (in a "`.gcr`" file), from which a paper character sheet may be printed. There is a similar data file for monsters, which is optimized to the purpose of giving the GM combat-related stats for each combatant in a convenient arrangement, as well as providing that data to the GMA software during game play.

By contrast the character record sheet is intended for characters who are expected to be part of the game for a much longer time and need much more detail recorded for them.

The GMA Monster encounter file (or "`.gmo`" file for short) holds the information for all the monsters encountered for the entire adventure. We don't

have a GUI yet to edit this file, so you will need to edit it using a text editor. Fortunately, this is a straightforward thing to do.

The basic file structure is hierarchical. At the top level, usually at the top of the file, is a stanza that describes the adventure module for which this `.gmo` file is to be used.

```
# Example encounter file for GMA Example module GX1
[module]
id=GX1
title=GMA Example Module
areas=A3

[area A3]
title=Goblin Chamber
monsters=goblin_trio
```

Here we see a few important items. As with the `.gcr` files discussed in Chapter 6, the file is organized into stanzas which are introduced by a bracketed name, within which are a set of data fields, one per line. Any line beginning with an octothorpe ("#", also known as a hash or pound sign) or semicolon (";") is a comment and is ignored by GMA.

The full technical description of the contents of a `.gmo` file appears in Appendix B, starting on page 433. The description of character record files starting on page 376 is also relevant.

The `[module]` stanza defines this file as pertaining to module `GX1` which is a name we arbitrarily made up for this example. This ID and the title of the module are for your reference only and can be whatever you like.

The `areas` field gives a list (separated by spaces) of areas within the module where we have creatures the PCs may encounter. In this file we have one area which we call `A3`, which is known as the "Goblin Chamber" in our module notes. Inside this area will be monsters identified in the file as "`goblin_trio`", which is our set of three goblins we've seen previously. Since GMA's core monster data includes goblins, this is a simple matter to list in the file in the definition for `goblin_trio`:

```
[m goblin_trio]
species=Goblin
random_qty=3
```

Each of these monster stanzas, named "`[m` ⟨*name*⟩`]`", defines a particular kind of monster or NPC. It may then appear in an "`[area` ⟨*areaname*⟩`]`" stanza's `monsters` field where a monster exactly matching the description for this kind of monster appears. If an identical set of monsters appears

elsewhere, that area may list the same monster as well. If we have an area B1 which also has a trio of goblins, we could have that area listed as

```
[area B1]
title=Guardroom
monsters=goblin_trio
```

and another set of three goblins would appear in that area, too.

This is the simplest possible example of a monster stanza. It simply specifies the species of creature (`Goblin`), which will retrieve the information about that kind of creature from GMA's core database. It then asks GMA to provide three individual monsters of that species with randomly-rolled hit points.

Printing and Using the Encounter Sheet

To use this `.gmo` file for an actual game, we can generate an encounter run sheet in PostScript using the `printmon` command, assuming we wrote it to a file called "`gx1.gmo`":

```
$ gma printmon gx1.gmo
WARNING: missing image_tag for monster 0
WARNING: missing image_tag for monster 1
WARNING: missing image_tag for monster 2
*** WARNING *** Duplicate name  in areas ['A3', 'A3', 'A3']
                                A3, A3, A3
```

Don't worry about the warning messages for now. They are warning us that we aren't taking advantage of some features of the `.gmo` file which will be useful later but we haven't discussed yet.

This will create a new file called `gx1.ps` which we can open in a document viewer program capable of viewing or printing PostScript files, or convert it to another form using a program like `ps2pdf`, and then print or view it using a program that understands that format.

```
$ ps2pdf gx1.ps gx1.pdf
```

This gives us a file as shown in Figure 7.1.

At the top of the stat block for goblins there are a few items which are highlighted in yellow. These are the values needed to enter the creature into the GMA initiative tracker, so they are highlighted to make it easy for the GM to spot them for quick reference. They are:

DEX Goblins have Dexterity scores of 15.

Pathfinder Encounter Run Sheet

GMA Example Module — MODULE ID: GX1

AREA: A3 — **Goblin Chamber**

Goblin — TYPE: humanoid (goblinoid) — IMAGE TAG: * — ALIGN: NE — REF: B156 — CR: 1/3 — XP: 135

This creature stands barely three feet tall, its scrawny, humanoid body dwarfed by its wide, ungainly head.

INIT	DEX	MOD	SIZE	SPC		RCH		HIT DICE	SPEED	BAB	CMB		CMD
+6	15	+4	S	5		5		1d10+1	30	+1	+0		12

S	D	C	I	W	C
11	15	12	10	9	6
+0	+2	+1	+0	−1	−2

FORT +3 — REFL +2 — WILL −1

DEFENSES

		= 10	+ ARMR	+ SHLD	+ DEX	+ SIZE	+ DODG	+ DEFL	+ NATL	+ MISC	DR	SR
AC	16	10	+2	+1	+2	+1						
Touch	13	10			+2	+1					RESIST/*IMMUNE	
Flat	14	10	+2	+1		+1					VULNERABLE	REGEN

SKILLS

Ride	+10
Stealth	+10
Swim	+4
Perception	−1

ATTACK DESCRIPTION	ATTACK	DAMAGE	THREAT	CRIT	RANGE	REACH
Sword, short	+2	1d4	19	×2		
Shortbow	+4	1d4	20	×3	10× 60′	

FEAT/SPECIAL ABILITY DESCRIPTION	REF
Darkvision 60′: Can see B/W in total darkness	562
Improved Initiative: «+4 to initiative»	127
... by class. Does breathe, eat, sleep	B308

FEAT/SPECIAL ABILITY DESCRIPTION	REF
Languages: Goblin	
Humanoid Type: prof. all simple or by class, own armor or	B308

NOTES

Goblins prefer to dwell in caves, amid large and dense thickets of thistles and brambles, or in structures built and then abandoned by others. Very few goblins have the drive to build structures of their own. Coastlines are favored, as goblins are quite fond of sifting through junk and flotsam in an unending quest to find treasures among the refuse of more civilized races.

Goblin hatred runs deep, and few things inspire their wrath more than gnomes (who have long fought against goblins), horses (who frighten goblins tremendously), and regular dogs (whom goblins regard as pale imitations of goblin dogs).

Goblins are also quite superstitious, and treat magic with a fawning mixture of awe and fear. They have the habit of ascribing magic to the mundane as well, with fire and writing both taking on mystical power in goblin society. Fire is much loved by goblins for its capacity to wreak great destruction and because it doesn't require size or strength to wield, but written words are hated. Goblins believe that writing steals words out of your head, and as a result of this belief, goblins are universally illiterate.

Goblins are voracious and can eat their body weight in food daily without growing fat. Goblin lairs always have numerous storerooms and larders. While they prefer human and gnome flesh, a goblin won't turn down any food−−except, perhaps, vegetables.

HIT POINTS FOR #1 (2) ☐☐

HIT POINTS FOR #2 (4) ☐☐☐☐

HIT POINTS FOR #3 (6) ☐☐☐☐☐☐

Figure 7.1: Encounter Run Sheet for GX1, Area A3

MOD They have an additional +4 to initiative in addition to that provided by Dexterity.

SIZE They are small creatures, so we use size code `S`.

C They have a Constitution score of 12.

There is one additional item, IMAGE TAG, which is unfamiliar to us at this point. It's also something `printmon` was complaining that we didn't specify, so let's rectify that before going further. The image tag for a creature is simply the name you would type into the mapper to identify the creature (and to provide the basis for finding an image to display as the creature token).

Previously, we were typing "`Goblin #1-3`" into the mapper. Since the `.gmo` file already specifies that we want 3 goblins, the "`#1-3`" part of that will be supplied for us automatically. We just need to specify the name part here. We'll do that by adding an `image_tag` field to our monster stanza.

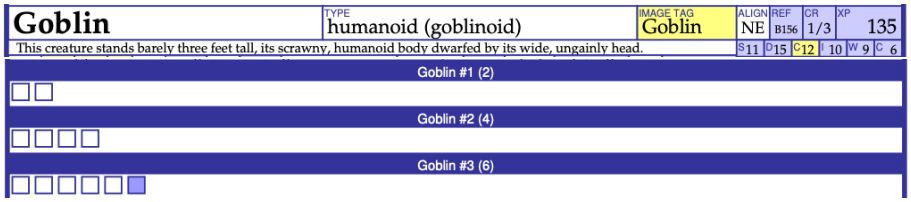
```
[m goblin_trio]
species=Goblin
random_qty=3
image_tag=Goblin
```

Running it through `printmon` again, we see:

```
$ gma printmon gx1.gmo
**WARNING** MISSING SPAWN POINT: Goblin
Goblin #1                       A3
Goblin #2                       A3
Goblin #3                       A3
```

Again, don't worry about the remaining warning message. We'll fix that later in this chapter when we learn what spawn points are. But we're making progress—and see that we now have Goblin #1, Goblin #2, and Goblin #3 in area A3 of the dungeon, as expected.

Now we have an actual IMAGE TAG shown at the top of the entry, and actual creature names listed at the bottom:



For each of the three goblins, their hit points are listed next to their names (e.g., Goblin #1 has 2 hit points). There is also a row of checkboxes for each of them, so during the game the GM can easily just tick off the boxes as the goblins take damage in the fight.

The encounter sheet is organized into sections. At the top are the basic stats such as ability scores and saving throws. Next is the defense information such as armor class and the list of skill modifiers. Following that is the offense information listing each form of attack. In this section alternating background colors group together attacks that may be made together. A dark background is used for ranged attack forms.

For example, the goblins shown in Figure 7.1 have a single *mêlée* attack (short sword) and a single ranged attack (shortbow). On the other hand, this creature prefers a single slam attack or a single hit with its falchion as a single attack, but either 2 slams or a double strike with the falchion if it can make a full-round attack.

ATTACK DESCRIPTION	ATTACK	DAMAGE	THREAT	CRIT	RANGE	REACH
slam	**+15**	**1d8+6 <+1d6 fire>**	**20**	**×2**		
2 slams	**+15/+15**	**1d8+6 <+1d6 fire>**	**20**	**×2**		
mwk [falchion]	**+16**	**2d6+9**	**18**	**×2**		
mwk [falchion] (full)	**+16/+11**	**2d6+9**	**18**	**×2**		

For another example, the manticore has this attack block:

ATTACK DESCRIPTION	ATTACK	DAMAGE	THREAT	CRIT	RANGE	REACH
bite	**+10**	**1d8+5**	**20**	**×2**		
2 claws	**+10/+10**	**2d4+5**	**20**	**×2**		
claw	**+10**	**2d4+5**	**20**	**×2**		
4 spikes	**+8/+8/+8/+8**	**1d6+5**	**20**	**×2**	**1× 180′**	

Here we see three groupings: a full-round attack which includes a bite and 2 claw attacks, a single claw attack, and a ranged attack to fire 4 spikes.

Any feats and special abilities are listed after the attacks. Some of these are added automatically based on creature type.

Finally, the notes section gives some highlights from the monster's listing in the source material. We can also add some encounter-specific notes of our own. For example, let's add a few notes to our entry for the goblins:

```
[m goblin_trio]
species=Goblin
random_qty=3
image_tag=Goblin
notes:
   These goblins are guarding the tribe's treasury of shiny
   rocks and pickle barrels.
   |
   **Morale** They are fanatically protective of their
   pickle barrels and will fight to the death.
```

Since the `notes` field accepts multiple lines of data, we indent all lines after the first one to show that they are a continuation of the value of `notes` onto more lines of the file.

Now when we run `printmon` again and look at the notes, our custom addition appears at the bottom.

NOTES

Goblins prefer to dwell in caves, amid large and dense thickets of thistles and brambles, or in structures built and then abandoned by others. Very few goblins have the drive to build structures of their own. Coastlines are favored, as goblins are quite fond of sifting through junk and flotsam in an unending quest to find treasures among the refuse of more civilized races.

Goblin hatred runs deep, and few things inspire their wrath more than gnomes (who have long fought against goblins), horses (who frighten goblins tremendously), and regular dogs (whom goblins regard as pale imitations of goblin dogs).

Goblins are also quite superstitious, and treat magic with a fawning mixture of awe and fear. They have the habit of ascribing magic to the mundane as well, with fire and writing both taking on mystical power in goblin society. Fire is much loved by goblins for its capacity to wreak great destruction and because it doesn't require size or strength to wield, but written words are hated. Goblins believe that writing steals words out of your head, and as a result of this belief, goblins are universally illiterate.

Goblins are voracious and can eat their body weight in food daily without growing fat. Goblin lairs always have numerous storerooms and larders. While they prefer human and gnome flesh, a goblin won't turn down any food--except, perhaps, vegetables.

These goblins are guarding the tribe's treasury of shiny rocks and pickle barrels.

Morale They are fanatically protective of their pickle barrels and will fight to the death.

Incidentally, note how we used `**Morale**` to set the word "Morale" in boldface. The full set of markup symbols available are documented starting

on page 690. Since the `.gmo` file doesn't allow blank lines in the middle of a multi-line field like `notes`, a line containing only a vertical bar (`|`) indicates a blank line in the field.

Let's go ahead and add the second room of goblins we mentioned earlier:

```
# Example encounter file for GMA Example module GX1
[module]
id=GX1
title=GMA Example Module
areas=A3 B1

[area A3]
title=Goblin Chamber
monsters=goblin_trio

[area B1]
title=Guardroom
monsters=goblin_trio

[m goblin_trio]
species=Goblin
random_qty=3
image_tag=Goblin
notes:
   These goblins are guarding the tribe's treasury of shiny
   rocks and pickle barrels.
   |
   **Morale** They are fanatically protective of their
   pickle barrels and will fight to the death.
```

Now when we run `printmon` we see:

```
$ gma printmon gx1.gmo
**WARNING** MISSING SPAWN POINT: Goblin
**WARNING** MISSING SPAWN POINT: Goblin
*** WARNING *** Duplicate name Goblin #1 in areas ['A3', 'B1']
Goblin #1                      A3, B1
*** WARNING *** Duplicate name Goblin #2 in areas ['A3', 'B1']
Goblin #2                      A3, B1
*** WARNING *** Duplicate name Goblin #3 in areas ['A3', 'B1']
Goblin #3                      A3, B1
```

Now we're getting duplicate name warnings. Should we be concerned about this? Maybe. It means that in the same dungeon we have two goblins

called Goblin #1, another pair both called Goblin #2, and two more who both have the name Goblin #3. Like encountering yourself in certain paradox-phobic Science Fiction novels, this is only going to be a problem if these two sets of goblins ever end up in the same place at the same time, because the mapper needs the name of each creature to be unique. (Really, people need this too. It's very confusing to have Goblin #2 and Goblin #2 in the same encounter with the PCs.)[1]

It would probably be better to have the second set be Goblin #4, Goblin #5, and Goblin #6. We do this by changing the monster definitions, although unfortunately that means we can't use the same monster definition stanza for both areas.

```
# Example encounter file for GMA Example module GX1
[module]
id=GX1
title=GMA Example Module
areas=A3 B1

[area A3]
title=Goblin Chamber
monsters=a3_goblins

[area B1]
title=Guardroom
monsters=b1_goblins

[m a3_goblins]
species=Goblin
random_qty=3
image_tag=Goblin
notes:
   These goblins are guarding the tribe's treasury of shiny
   rocks and pickle barrels.
   |
   **Morale** They are fanatically protective of their
   pickle barrels and will fight to the death.

[m b1_goblins]
```

[1] Having said this, there *are* times when you do want that confusion, as when the party meets a set of doppelgängers imitating them. There's a nice dramatic moment to enjoy when Alice, Bob, and Charlie encounter another Alice, Bob, and Charlie on the map, and appear that way in the initiative tracker. Instructions for how to accomplish this start on p. 142, but note that you still need to have a unique name for each in the `.gmo` file, following the advice given here. There's just a way to make them *appear* to be duplicates from the players' points of view.

```
species=Goblin
random_qty=3
image_tag=Goblin#4
```

Adding the "#4" to the end of the `image_tag` of the second group tells GMA where to start numbering this set of goblins from. Now we run `printmon` on it.

```
$ gma printmon gx1.gmo
**WARNING** MISSING SPAWN POINT: Goblin
**WARNING** MISSING SPAWN POINT: Goblin#4
Goblin #1                      A3
Goblin #2                      A3
Goblin #3                      A3
Goblin #4                      B1
Goblin #5                      B1
Goblin #6                      B1
```

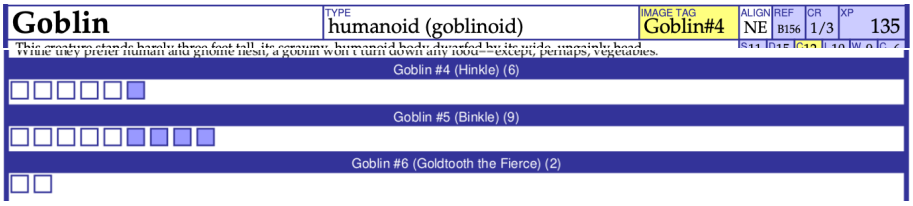
Individuals and Groups

What we've seen so far is useful for making nameless hordes of copies of a single kind of creature. We may want to individualize them a bit, however, such as giving them each their own name, or determining hit points for them rather than having them randomly determined by GMA.

To give them individual names, we'll use the `name` field. Since names can have more than one word, this field expects one monster's name per line. Each line after the first is, like all multi-line fields in a `.gmo` file, indented to show that it's part of the same field.

```
[m b1_goblins]
species=Goblin
random_qty=3
image_tag=Goblin#4
name:Hinkle
     Binkle
     Goldtooth the Fierce
```

```
$ gma printmon gx1.gmo
**WARNING** MISSING SPAWN POINT: Goblin
**WARNING** MISSING SPAWN POINT: Goblin#4
Goblin #1                       A3
Goblin #2                       A3
Goblin #3                       A3
Goblin #4                       B1
Goblin #5                       B1
Goblin #6                       B1
```



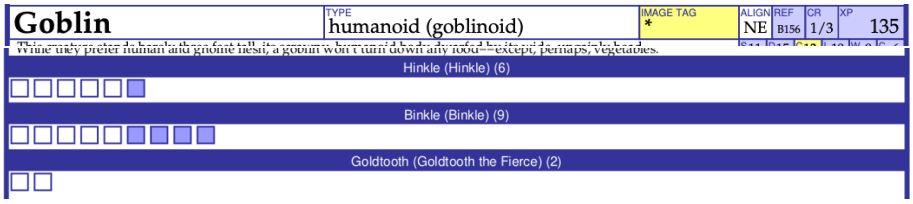
Now we have individual names for these three, but the image tag is still reading as Goblin #4, Goblin #5, and Goblin #6, so that's what we're expecting the mapper to show for their tokens (which also means each will still get the stock "Goblin" image).

Looking at the individual section, we see that since GMA knows that we customized things a bit, it's displaying each monster's image tag and name along with their hit points.

However, what if we want the tokens on the map to show their names rather than just "Goblin #4" since they now have names of their own? We simply put their names in the `image_tag` field. This field may have multiple tags, separated by spaces, if each individual has their own tag.

```
[m b1_goblins]
species=Goblin
random_qty=3
image_tag=Hinkle Binkle Goldtooth
name:Hinkle
     Binkle
     Goldtooth the Fierce
```

```
$ gma printmon gx1.gmo
**WARNING** MISSING SPAWN POINT: Goblin
**WARNING** MISSING SPAWN POINT: Hinkle Binkle Goldtooth
Binkle                          B1
Goblin #1                       A3
Goblin #2                       A3
Goblin #3                       A3
Goldtooth                       B1
Hinkle                          B1
```



Now we see that on the sheet, the IMAGE TAG area just has a "*" in it to show that the image tag varies with the individuals, so you should just look at the lower section to find them. And there we see that the individual tags are indeed there. So on the mapper, you would add these creatures as "`Hinkle`", "`Binkle`", and "`Goldtooth`".[2]

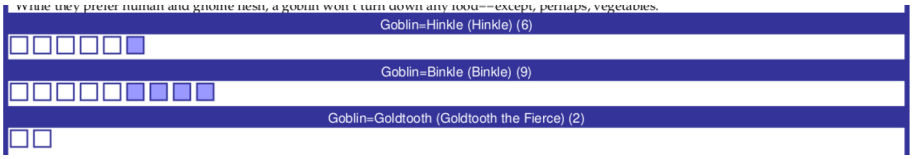
This also means the mapper will expect an image to be defined for each of them, called `#Hinkle`, `#Binkle`, and `#Goldtooth`.[3] That's fine if you want to have a different image for each of them. But maybe you just want each of their tokens to be labelled with their name but have the common `Goblin` image. No problem. The `image_tag` field is flexible enough to accommodate a number of special cases like this. For this, just put the name of the image to use, an equals sign (=), and the creature's name as it should appear on the label. So we change the `image_tag` field to read:

```
image_tag=Goblin=Hinkle Goblin=Binkle Goblin=Goldtooth
```

This time, the image tag for each goblin shows that, and you can type it that way into the mapper as well.

[2]You may be asking yourself, "If I'm typing this into the mapper, why can't I just type whatever I want without having it also be in the `.gmo` file? Two reasons: (a) it leads to less confusion for you if the same names are shown everywhere you look in your notes, and (b) when we get to Chapter 9 we'll see how this all becomes automated so GMA will put the creature tokens on the mapper for you, so it really does need to know what you want it to put there.

[3]Or perhaps `^0^Hinkle`, `^0^Binkle`, and `^0^Goldtooth`, if you're so inclined, along with `%Hinkle`, etc. But I digress.



(We could also have omitted the `name` field and just use the `image_tag` alone if we didn't otherwise need full names listed for each goblin.)

What happens if there are more monsters than image tags? The last tag is repeated for all the remaining monsters. Let's change the encounter to have four goblins but leave the `image_tag` field alone:

```
random_qty=4
image_tag=Goblin=Hinkle Goblin=Binkle Goblin=Goldtooth
```

We see that this repeats the last tag, `Goblin=Goldtooth`, for the extra goblin. GMA automatically added a sequence number to them, so we now have Goldtooth #1 and Goldtooth #2.



Random Name Generation

We can also have GMA generate random names for our monsters. To do this, we need to specify what *culture* to generate names for. A culture, in this sense, is simply a defined pattern of what names from that culture would look like. To see the list of defined cultures, run the `namegen` command shown in Fig. 7.2. Let's say the goblins have Chelaxian names for whatever reason. Let's find out what name generation options the Chelaxian culture gives us:

```
$ gma namegen --describe --culture chelaxian
Culture:           chelaxian
Includes Surnames? yes
Genders:           F, M, S
Default lmin (F):  4
Default lmax (F):  15
Default lmin (M):  3
Default lmax (M):  12
Default lmin (S):  4
Default lmax (S):  12
```

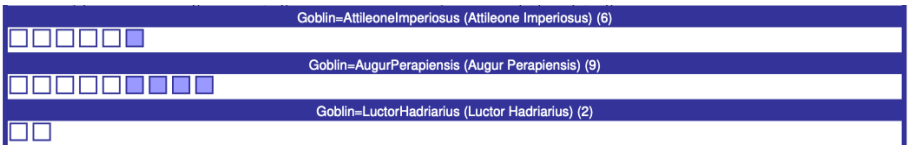
```
$ gma namegen --list-cultures
Cultures available (specify to --culture/-c option):
  azlanti
  bekyar
  bonuwat
  chelaxian
  dwarf
  elf
  erutaki
  garundi
  gnome
  half-orc
  halfling
  keleshite
  kellid
  kitsune
  shoanti
  taldan
  tian-dan
  tian-dtang
  tian-hwan
  tian-la
  tian-min
  tian-shu
  tian-sing
  ulfen
  varisian
  vudrani
  zenj
```

Figure 7.2: Cultures Available for Name Generation

The significant parts of this are that this culture includes surnames (otherwise we could only generate given names, but with this one we have the option to generate full names too), and the *genders* available are F, M, and S. In this context, gender refers only to the fact that there's a set of distinct naming patterns we can choose from. Gender S stands for surnames, and M and F are, generally speaking, for male and female names from that culture (but they need not be, and there could be many genders to cover name differences related to anything at all).

Since we're generating the names, we don't specify the `name` field anymore. We just use the `image_tag` field. In place of the name of each creature, we will put `@chelaxian,M` to generate a male Chelaxian name, and `@chelaxian,F` to generate a female one.

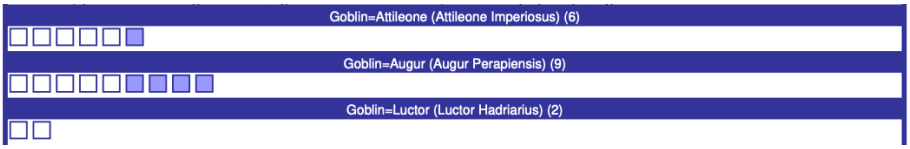
```
species=Goblin
random_qty=3
image_tag=Goblin=@chelaxian,M Goblin=@chelaxian,F
    Goblin=@chelaxian,M
```



Note that since the Chelaxian culture includes surnames, we got full names for each. This auto-generates the `name` of each creature to be the full name (e.g., "Attileone Imperiosus"), and that name is compressed into the image tag (e.g., "`Goblin=AttileoneImperiosus`").

That may be a little cluttered for the creature token label on the map, so we can include the `if:1` option (this stands for "image first name", and if the value after the colon is a 1, it enables a feature to strip away all but the first name from the image tag). This goes between the culture name and before the gender.

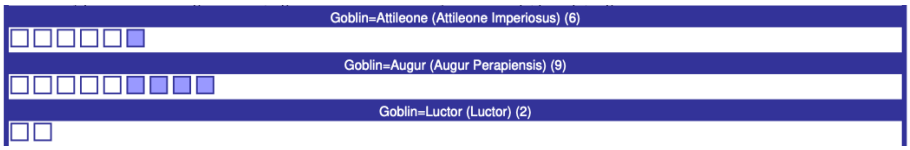
```
species=Goblin
random_qty=3
image_tag=Goblin=@chelaxian,if:1,M Goblin=@chelaxian,if:1,F
    Goblin=@chelaxian,if:1,M
```



Now the image tags for the mapper are shorter (e.g., "`Goblin=Attileone`", which will display simply as "Attileone" on the token with the Goblin graphic image).

We can also just turn off surnames entirely by including the `surnames:0` option. Let's say the third goblin has no surname at all.

```
species=Goblin
random_qty=3
image_tag=Goblin=@chelaxian,if:1,M Goblin=@chelaxian,if:1,F
    Goblin=@chelaxian,surnames:0,M
```



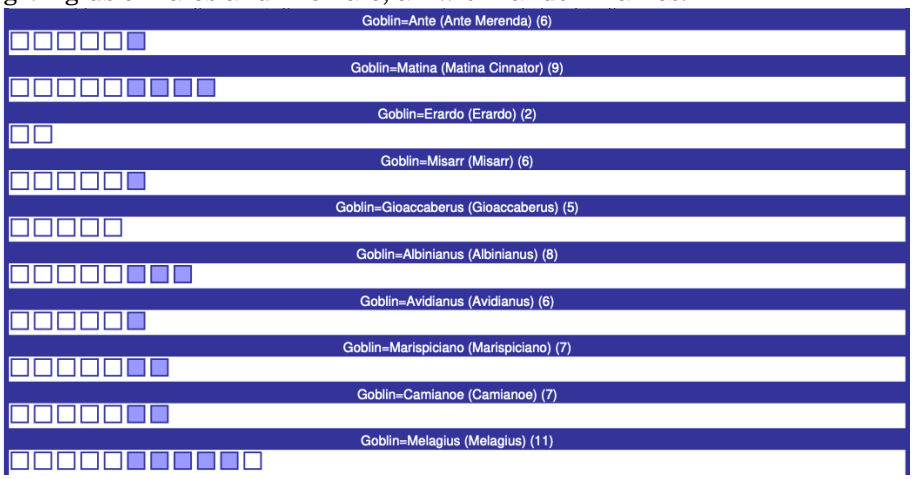
If we want everyone in this area to have Chelaxian names, and to only use their first names in image tags, we can specify it globally at the area level:

```
[area B1]
title=Guardroom
monsters=b1_goblins
culture=chelaxian,if:1
```

Now we can just list the genders (or any creature-specific options) in the `image_tag` field:

```
[m b1_goblins]
species=Goblin
random_qty=10
image_tag=Goblin=@M Goblin=@F Goblin=@surnames:0,M
```

Since the last image tag repeats, if we set the number of goblins to 10, as we did above, the tag `Goblin=@surnames:0,M` will repeat for all of the rest, giving us 9 males and 1 female, all with random names.



We can even define the global culture setting at the module level.

```
[module]
id=GX1
title=GMA Example Module
areas=A3 B1
culture=chelaxian,if:1
```

See `gma-monster-sheet`(5) starting on page 433 in Appendix C for the full details about specifying culture options in `.gmo` files and the entry for `pfnamegenerator`(3) in Appendix B (page 306) which discusses random name generation in depth.

Generic Creature Names

We have seen how we can list specific species names in our encounter file for each creature, or even actual given names for them. However, it's also quite likely that putting that information on the mapper client and initiative order list would provide information the players don't yet know about the

encounter. Without a formal introduction, they wouldn't know that a goblin was named Avidianus, so providing that information may be distracting enough to remove the player from some of their immersion.

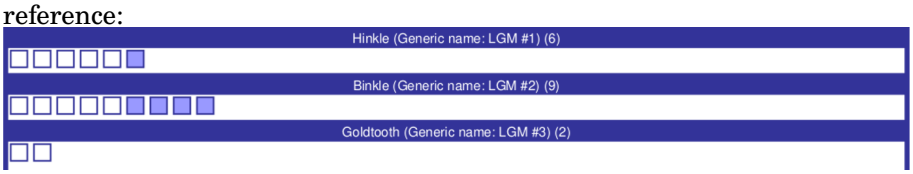
If the players haven't yet encountered goblins, it may even be too much to reveal that the creatures they just met are goblins rather than something more generic like "little green men."

To facilitate better gameplay, GMA allows for a generic name (like "little green men" or "LGM" for short) to be used everywhere visible to the players until they find out the details. Later on, of course, we can switch to the more specific names as they become known to the players.

In the `.gmo` file, we add a `generic_name` field to the monster definition, like so:

```
[m b1_goblins]
species=Goblin
random_qty=3
image_tag=Hinkle Binkle Goldtooth
generic_name=LGM
```

On the encounter sheet, the generic name for each is shown for the GM's reference:

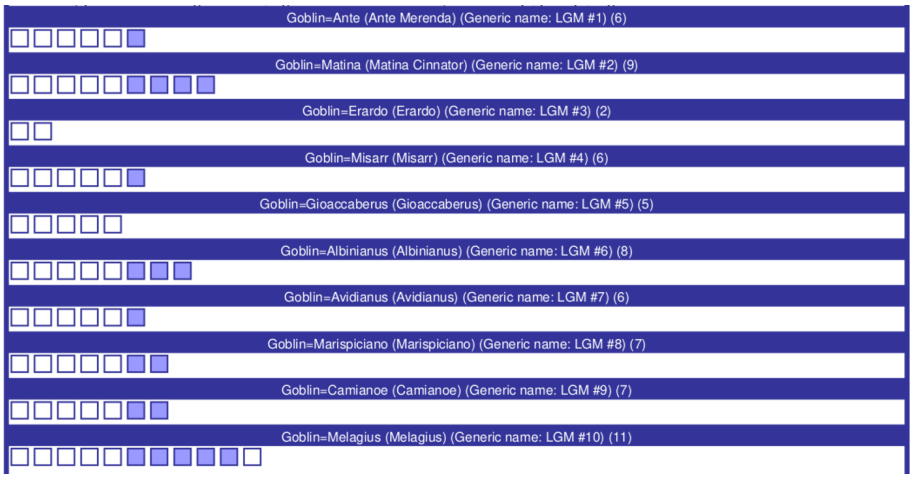


In our larger example with the ten randomly-named goblins, it would look like this:

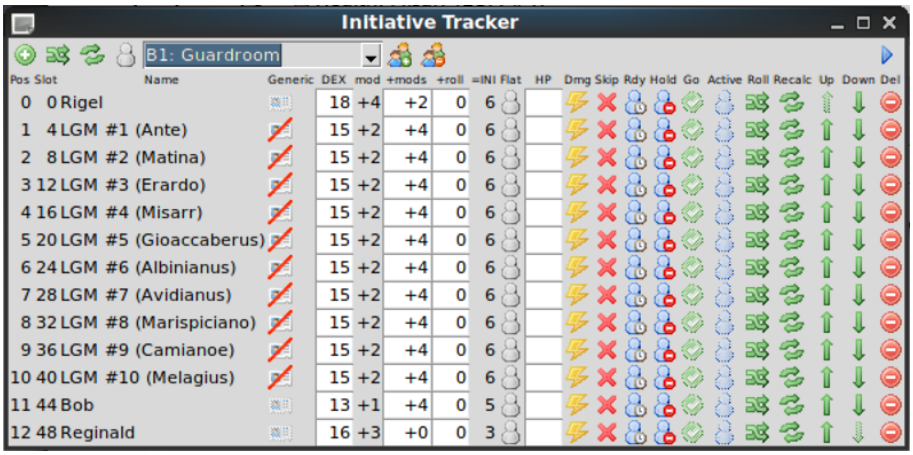
```
[m b1_goblins]
species=Goblin
random_qty=10
image_tag=Goblin=@M Goblin=@F Goblin=@surnames:0,M
generic_name=LGM
```

Generic names are sequenced separately from image tags. If multiple creatures are defined at once, the suffixes "`#1`," "`#2`," and so forth are automatically added. If you want to begin these suffixes with a different number, just append it to the end of the generic name without an intervening space, such as

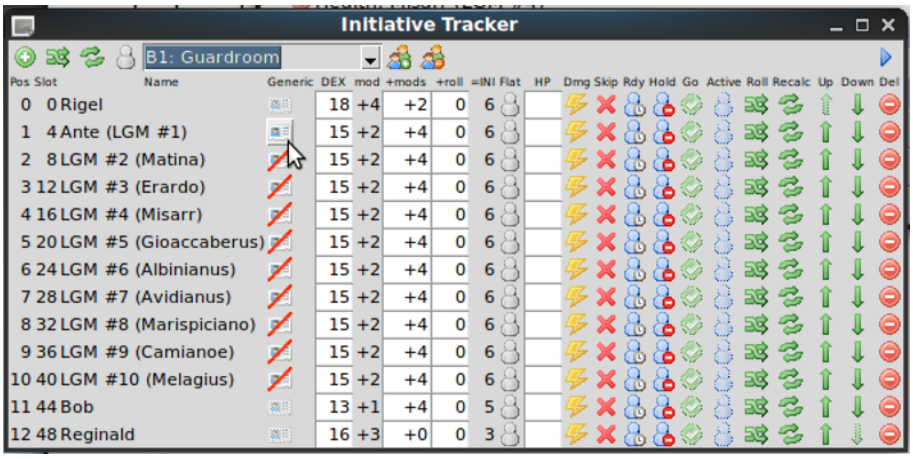
```
generic_name=LGM#5
```



In GMA's initiative tracker, creatures who have generic names will display the current name (real or generic) being displayed to the players first, followed by the other one in parentheses. There is also a "name tag" button next to the name, which has a slash through it if the generic name is being displayed (i.e., "not" the real name tag):



When the party learns that LGM #1's actual name is Ante, the GM clicks on the name tag ( ) button. The name for that creature switches to the real name, as shown in the tracker and by the removal of the slash through the name tag icon. Clicking the same button again will swap the names back.



How Do Names Really Work?

So far we have seen a number of fields which influence what the creature will be known as, each in different ways: `biased_qty`, `generic_name`, `hp`, `image_tag`, `maximized_qty`, `median_qty`, `name`, `random_qty`,

But with the large number of ways these can be customized it may not yet be clear just how they interact with each other to determine the ultimate name for the creature, as some take precedence over others.

Here's how GMA interprets these values when reading the `.gmo` file:

1. The `biased_qty`, `hp`, `maximized_qty`, `median_qty`, and `random_qty` together determine the number of creatures to be created. If there are more than one, GMA will add "`#1`", "`#2`", "`#3`", and so forth to the end of each name to differentiate them. If the supplied creature name already ends in a number (e.g. "`Goblin#4`"), then the sequence will start counting with that number instead of 1.

2. If `generic_name` is defined, that will be used on the map until the GM decides to reveal the creature's individual name (as determined below).

3. If the `image_tag` contains an "`@`" character, it is interpreted as a generated creature name according to the culture settings currently in effect. The generated name is used as if it had been explicitly written into the `image_tag`.

4. If `name` is defined, it is only used to denote individual names in the paper encounter sheets.

Duplicate Nametags

Normally, every creature token on the map must have a unique name, so that there is no confusion as to what to do about two creature tokens both named "Alice" on the board at the same time. (This is also necessary for the internal workings of the mapper and the GMA GM tools which control the monsters during combat, so each creature can be accurately identified.)

However, the nature of these games being what it is, sometimes you *do* in fact want that kind of confusion. What better way, after all, to have players confront that classic dilemma of facing a shapeshifting villain who has taken the form of their captive, than by having them enter a room to rescue Bob only to see two identical creatures, both labelled as "Bob"?

We can accomplish this by taking advantage of a naming convention built into the mapper by naming the disguised villain "`false_Bob`". This way each does have a unique name. The two creatures will appear as "`Bob`" and "`false_Bob`" on all GM tools, including the initiative tracker. However, the mapper client will omit all text leading up to an underscore ("_") from everything shown to the map user. This means that `false_Bob` will display as `Bob` in its creature token name as well as in menus and other output.

A Random Comment About Randomness

If you were paying close attention, you may nave noticed that the randomly-generated names and hit points for the goblins gave consistent results no matter how many times we ran `printmon`—almost. In the final example, it looks like we got different names for the goblins.

GMA's random die-roller that's built in to the monster encounter generator has two contradictory goals: provide randomly-determined values for things like monster hit points and names, but to then always roll the same random values every time, so you don't get a different name and hit point total for every creature every time you reprint the sheet or load the data into GMA for another session.

To accomplish this, GMA bases the random number sequences on the module, area, and creature in a way that will always yield the same results for the same creature, as long as too much about the creature doesn't change. So keep in mind that once you set up the encounters in the `.gmo` file, everything should stay stable across multiple reloads of that data file, but making certain changes in the `.gmo` file (other than adding additional areas and their denizens to the file) might reshuffle the random parts of the creatures you're changing.

Hit Points

There are a few options you can choose as alternatives to simply rolling random hit points for each creature. If you want to determine the hit points

for your creatures yourself, you can enter them in the `hp` field instead of using `random_qty`.

```
[m b1_goblins]
species=Goblin
image_tag=Goblin=@M
hp=12 11 14
```

This will generate three goblins, with 12, 11, and 14 hit points respectively.

If you just want *some* of them to have random hit points, place an `@` in place of a number to have that one creature randomly generated.

```
[m b1_goblins]
species=Goblin
image_tag=Goblin=@M
hp=12 @ 14
```

In this case, the 2nd goblin in the set will have random `HP` while the other two are fixed.

You can use both `hp` and `random_qty`. The latter effectively adds a number of `@`s to the hit point list, so

```
hp=12 @ @
```

and

```
hp=12
random_qty=2
```

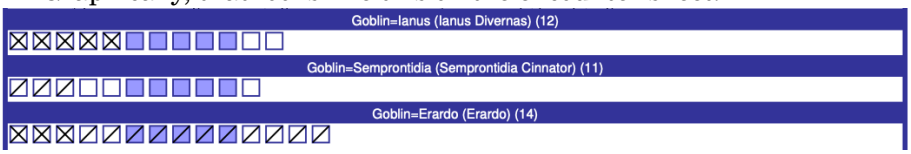
are equivalent.

If you want to indicate that a creature has already sustained damage, you can specify that by adding the lethal and non-lethal wounds to the total hit point number.

```
hp=12:5 11:0:3 14:3:X
```

This says that the first goblin has 12 hit points but has suffered 5 points of damage, the second has 11 hit points and 3 points of non-lethal damage, while the third has 14 hit points, has lost 3 of them due to lethal damage, and all the remaining points to non-lethal damage.

Graphically, that looks like this on the encounter sheet:



You can use an `X` in place of either the lethal or non-lethal number to indicate that all remaining points are used up.

Alternatively, if you use `!` instead of the total hit point number, like this:

```
hp=12 ! 14
```

Then that creature has the maximum possible number of hit points—as if all dice came up with their highest values.

If you want to use median values for monster hit points, use a `~` character instead of the number of hit points. For example:

```
hp=~
```

This will assume each die rolled its median number (e.g., a d8 would roll a 4).

Advanced Customization

So far we have used the goblin species as-is from the GMA core data. Suppose we want to use that as a basis, but give some goblins different weapons, armor, or maybe even some class levels or other improvements?

This is easily done. When you specify `species=Goblin`, that starts with the information for goblins, but we can override the standard values for any aspect of the creature by adding more data fields with the updated information.

Applying standard templates such as Advanced is now supported by GMA, but currently the number of supported templates is limited. More will be added over time or upon request.

For example, if we wanted to change the goblins to lawful evil outsider goblins from another plane of existence, and change their hit dice to 2d8+2, we could specify this as

```
[m b1_goblins]
species=Goblin
type=humanoid (goblinoid, outsider)
```

```
dice=2d8+2
align=LE
```

We can turn humans into skeletons by applying the `skeleton` template:

```
[m skeletons]
species=Human
class=Human Skeletons
templates=skeleton
random_qty=5
```

For very specific NPCs who have full class levels and other customization, you will essentially create the entire character in the `.gmo` file as documented in detail on pages 396–418. For making monsters fully described in the `.gmo` file (and thus not based on any particular species in the core data), you should use the field

```
species=_BaseMonster
```

to indicate that we're starting with an entirely blank canvas with this creature, not another existing species.

One thing to keep in mind is that the focus of monster entries is to represent a creature's stats which may vary from the stock creature type with minimal overhead or complication, as opposed to the character record sheet's focus of maintaining a complex PC over time, adhering to standard character configuration rules. The upshot of this is that attributes such as armor class, saving throws, attack and damage values, etc., are used as entered in the `.gmo` file. There is no calculation happening behind the scenes such as automatically adding dexterity modifiers to armor class. Each attribute needs to be completely specified.

For example, if a creature's stat block includes an entry such as "**AC** 16, touch 13, flat-footed 14 (+2 armor, +2 Dex, +1 shield, +1 size)," even though the `.gmo` file and/or core data it references may know the creature is small size and has a 15 Dex, the armor class would still be listed in the `.gmo` file as

```
ac=armr|+2
   dex|+2
   shld|+1
   size|+1
```

to specify the values from which the armor class (including flat-footed and touch) are calculated by GMA while processing the `.gmo` file.

Attack and Damage Roll Syntax

For reasons which will make more sense in Chapter 9, some special care needs to be taken around how attack and damage are listed in the `.gmo` file for a creature.

For example, the attack block for a creature might read

```
attacks=@clear
    0|Rapier|M|name:+1 rapier,attack:+10,damage:1d6+6|no
    1|_CreatureAttackForm|M|name:bite,attack:+9,damage:1d6+1
+ 1d4 acid|no
    2|CrossbowLight|M|name:mwk light crossbow,attack:+5,damag
e:1d8 + poison|yes
```

This produces an attack block that looks like this:

ATTACK DESCRIPTION	ATTACK	DAMAGE	THREAT	CRIT	RANGE	REACH
+1 rapier	+10	1d6+6	18	×2		
bite	+9	1d6+1 + 1d4 acid	20	×2		
mwk light crossbow	+5	1d8 + poison	19	×2	10× 80′	

As we will see as we automate GMA's interaction with the mapper, the players will have visibility of the ATTACK DESCRIPTION, ATTACK, and DAMAGE fields during combat, and the automated die roller will be available to roll the attack and damage amounts for us based on the contents of the latter two of those fields.

Since we don't want to give away the fact that the monster's rapier is a +1 magic weapon, or even that the crossbow is masterwork quality, we will enclose the parts we want the players to see in square brackets if we don't want them to see the entire field:

```
attacks=@clear
    0|Rapier|M|name:+1 [rapier],attack:+10,damage:1d6+6|no
    1|_CreatureAttackForm|M|name:bite,attack:+9,damage:1d6+1
+ 1d4 acid|no
    2|CrossbowLight|M|name:mwk light [crossbow],attack:+5,dam
age:1d8 + poison|yes
```

Also, to avoid confusing the die roller, if there are extra comments like "+ poison" that are understandable to a human reading the sheet but which won't make sense to GMA, we likewise put brackets around the portion the die roller will need to consider. It will ignore the rest of the field:

```
attacks=@clear
    0|Rapier|M|name:+1 [rapier],attack:+10,damage:1d6+6|no
    1|_CreatureAttackForm|M|name:bite,attack:+9,damage:1d6+1
+ 1d4 acid|no
    2|CrossbowLight|M|name:mwk light [crossbow],attack:+5,dam
age:[1d8] + poison|yes
```

(The die roller can cope just fine with "`1d6+1 + 1d4 acid`" but not with "`+ poison`".)

Finally, if there is part of a damage roll which should not be multiplied on a critical hit, we need to enclose that part in angle brackets. That would include the acid damage from the creature's bite attack:

```
attacks=@clear
    0|Rapier|M|name:+1 [rapier],attack:+10,damage:1d6+6|no
    1|_CreatureAttackForm|M|name:bite,attack:+9,damage:1d6+1
<+ 1d4 acid>|no
    2|CrossbowLight|M|name:mwk light [crossbow],attack:+5,dam
age:[1d8] + poison|yes
```

Enclose the entire part that should be excluded from being multiplied, including the math operators that connect it to the other parts of the damage roll, so that when the part in angle brackets is removed, the remaining text still makes a valid die roll specification.

Polymorphing and Shapechanging

Some creatures have the ability to alter their form, leading us to want to have multiple images for their creature token. Then we can change their appearance on the mapper to match how they look at any given time.

If we have a creature which can have four different shapes, one that is medium-size, another that is large, and two more that are huge and gargantuan, we list the additional sizes in the `size` field, separated by spaces. The number of "skins" the creature can shape-change into is the number of sizes listed here, so it's quite possible for some or all of them to be the same size.

```
size=M L H G
```

(In previous versions of GMA, there was a separate `poly_sizes` field for this, but now the `size` field covers this case instead.) This causes the following information to be placed in the monster's feat block:

FEAT/SPECIAL ABILITY DESCRIPTION	REF
Sizes: 0=M, 1=L, 2=H, 3=G	

More importantly, tools like the mapper will know that the creature has four different forms, and what sizes those forms take The mapper will expect to find four different images for the creature, corresponding to these four different *skins*. These will be in files called `^0^`⟨*name*⟩, `^1^`⟨*name*⟩, `^2^`⟨*name*⟩, and `^3^`⟨*name*⟩ (or the shorthand forms `#`⟨*name*⟩, `=`⟨*name*⟩, `-`⟨*name*⟩, and `+`⟨*name*⟩) as described in Chapter 5.

If the polymorphing or shapechanging creature has multiple disguises or appearances that are the same size, the `size` field would look something like this:

```
size=M M M M
```

That's fine as far as it goes for telling the software that the creature has four different token images, but it's not very helpful for the GM to be able to remember which one is which during the game. To help with this, you can also add a comment string to the end of any size code, separated from the code by a colon, such as:

```
size=M:base M:goblin M:orc M:wizard
```

Use of the `size` field supersedes the older notation of adding "+⟨*n*⟩" to the end of the `image_tag` to indicate that there were ⟨*n*⟩ additional forms for the creature. You can still do that as well, if you like, but the only effect it has is to show that notation in the IMAGE TAG field of the paper form.

The default skin GMA will use when adding the creature token to the map is the first one in the `size` field ("`M:base`" in the example above).

If you want one of the other skins to be used for the creature's first appearance, place an asterisk after the colon in its comment. For example:

```
size=M:base M:goblin M:*orc M:wizard
```

in which case the third skin ("orc") will be its initial appearance on the map.

Normally, if a creature can polymorph (or use different disguises, etc.), the context menu in the mapper will show a Polymorph . . . item for that creature. It may be the case, however, that the fact that the creature can shapechange—or even the details of what the different skins are—and the ability to control them should be withheld from the players. In this case, add the field

```
poly_gm=yes
```

to that creature's entry in the `.gmo` file. This will instruct the mapper to activate the Polymorph ... menu only for the GM.

Custom Sizes

A size category letter may be specified for the `size` field of a monster entry in a `.gmo` file to give the creature a size of your choice. This, and the values specified in the `poly_sizes` list, may be any of the standard size category codes: `F` (fine), `D` (diminutive), `T` (tiny), `S` (small), `M` (medium), `L` (large), `H` (huge), `G` (gargantuan), and `C` (colossal). To indicate a creature that is "tall" or "long", use an upper-case or lower-case letter respectively.

In most cases, this will suffice, but sometimes a creature needs a non-standard reach distance. To indicate this, follow the size category with the number of feet of natural reach for the creature. For example, a size value of "`M20`" indicates a medium creature with a 5' space but a 20' reach instead of the normal 5' reach.

Each creature has an extended reach zone of double their natural reach. This is usually used when wielding a reach weapon appropriate for their size. A custom extended reach distance may be given by adding "`->`" and the extended distance. For example, a size value of "`S10->25`" indicates a small creature with a 10' natural and 25' extended reach, while "`M->20`" indicates a medium creature with the normal 5' natural reach and a 20' extended reach.

In the rare case when a non-standard creature space is needed, add "`=`" followed by the space in feet to the end of any of the above forms. This is not supported for making creatures larger than the standard sizes, not smaller. For example, the Oliphaunt of Jandelay has a space of 80' and a natural reach 80' beyond that, so its size code would be "`c80=80`".

Initial Monster Conditions

Sometimes a monster enters the battlefield with conditions already in place. For example, zombies are always considered to be staggered in Pathfinder. To do this, simply add a `conditions` field to the monster definition.

This is formatted similarly to the `feats` field, in that it is a multi-line field with one condition listed on each line. Conditions may have spaces in their names.

If the special value "`@clear`" appears in the condition list, all previously-specified conditions are cleared. Otherwise, the values here add to those which may already have been present in the base monster definition in the GMA database.

To set a creature to have the blinded and staggered condition as soon as it is spawned into the game, add a field like this:

```
conditions:
   blinded
   staggered
```

These condition names must match those defined for your game server (i.e., the same ones available in the condition menus for GMA and the mapper client).

Permanent Invisibility

If you place parentheses around a creature's spawnpoint (e.g., "`spawn=(B2)`" rather than "`spawn=B2`"), that creature starts out hidden from view as well as inactive (skipped) on the initiative list. This means that the players will see no indication of it. The creature token only appears on the GM mapper client. Once you activate the creature in combat, its token will become visible to the players.

But what if the creature wasn't merely out of sight until that point, but was truly invisible, such that you want its creature token to remain completely hidden from the players? To accomplish this, just add a field to the monster definition:

```
hidden=yes
```

This will stop GMA from ever revealing the token to the players' map clients. When you wish to reveal it, simply right-click on the token on the GM mapper client and de-select the "hidden" condition for it.

Basing One Monster On Another

It is often the case that the same sort of monster (such as, for example, a hobgoblin guard) will appear in more than one encounter area. In cases where the monster is a stock creature taken out of the GMA core data, it is a simple matter of setting up a monster definition for each encounter that has a `species` field along with the specific settings for that area, such as `name`, `random_qty`, etc.

However, if a monster is defined entirely in the `.gmo` file, or is heavily customized beyond the base species description from the core data, it is less convenient to have to repeat all the customization fields for each area where that customized creature is found.

To help with this, the `.gmo` file format includes a `based_on` field. If a monster entry looks like this:

```
[m guards9]
based_on=hobgoblinguard
random_qty=3
image_tag=fighter::HobFtr=Hobgoblin
spawn=AN16 AN17 AP16
```

GMA will understand that the three monsters known as `guards9` should be the same as the (presumably highly customized) monster defined in stanza `[m hobgoblinguard]` but with the `random_qty`, `image_tag`, and `spawn` fields given here rather than the ones appearing in the definition of `hobgoblinguard`.

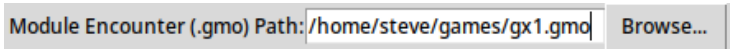
If a monster definition stanza includes a `based_on` field, GMA will first read all of the fields from the named monster stanza, and then any other fields in this stanza will override what was in the stanza this was based on. (If you are familiar with programming in C or C++, think of `based_on` just as you would the `#include` statement in those languages.)

The referenced monster stanza may itself have a `based_on` field which pulls from yet another monster definition stanza, and so on, allowing for a great deal of flexibility in defining and customizing monsters in the `.gmo` file without needlessly repeating information.

Automating Combat Setup with the GM Console

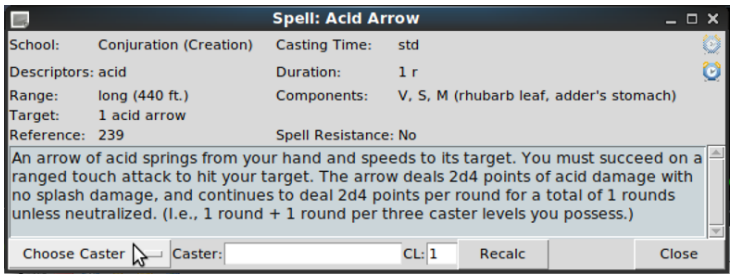
Remember all the effort we went to back in Chapter 3 to add all the combatants to the initiative tracker? We don't have to do that anymore, now that we have prepared our encounters in the `gx1.gmo` file.

We just need to enter the pathname of the `gx1.gmo` file in the campaign preferences settings.

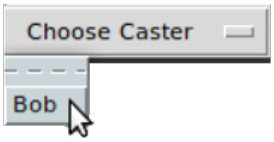


With this setting in place, GMA will load our `gx1.gmo` encounter file every time it starts. We can also reload it if needed by selecting Module > Reload current module from the menu.

We also need to be sure to list the PCs and any NPC allies who should always be pre-loaded into the initiative list, as well as the caster levels for any party spellcasters, in the campaign world preferences. One of the reasons to do so is so that the display of spell information can be scaled to a particular caster. For example:



Picking Bob from the list, we see that his name and caster level are placed into those fields.





Now let's look at using the encounter automation features. Before we can do this, we need to fix the remaining issue we have been ignoring up to this point with our `.gmo` file—the spawn point for each monster. In each monster stanza we need to have a `spawn` field which gives the map coordinates of where the monster token should be placed when the creature is encountered.

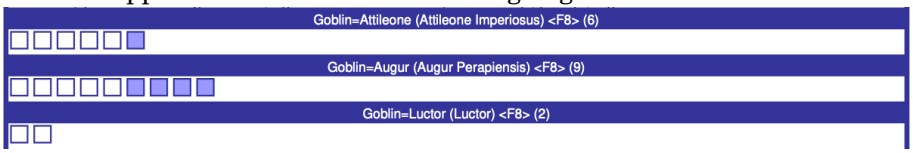
The coordinates are given using the letter-number labels shown on the mapper itself:

```
spawn=C12
```

Assuming we changed the goblins in the guardroom to be:

```
[m b1_goblins]
species=Goblin
random_qty=3
image_tag=Goblin=@M Goblin=@F Goblin=@surnames:0,M
spawn=F8
```

then these coordinates are printed on the sheet to remind us where to click on the mapper to add these tokens when going into the encounter.



The default behavior is for the first creature in the set to be at the designated point, and each one after that to be five feet to the right, so the three goblins generated here would be located at coordinates F8, F9, and F10. (We will see later when we connect the GM console and the mapper over a network, that the mapper will acutally place them for us in these locations; if more than a few creatures are placed, it will arrange them in a group with the upper-left corner at the spawn point to avoid making a long line to the right of the first one.)

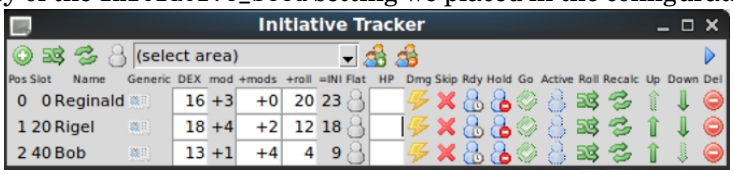
If we want to specify different spawn points for each creature, we simply give more coordinates, separated by spaces:

```
spawn=F8 G5 H7
```

Although most monsters are encountered in one location, you may have a recurring villain or the like, who appears in more than one location during the adventure. If you mention the same creature in more than one area, you can give specific spawn points for each area. Say a creature appears at map coordinates AA12 if encountered in area E1, but at coordinates X15 if encountered in area Tower7. In this case, the `spawn` line for that creature would look like this:

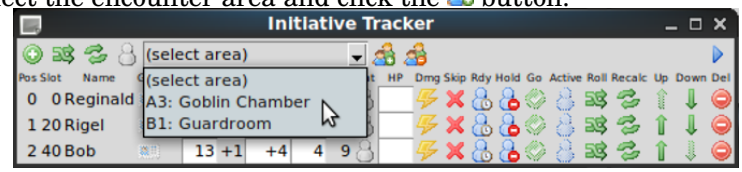
```
spawn=E1@AA12 Tower7@X15
```

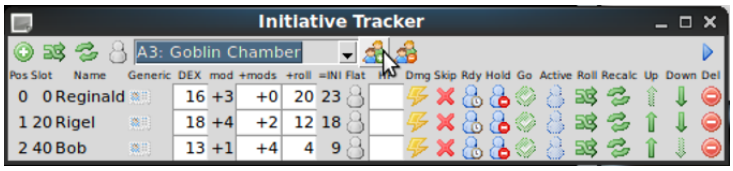
Now that we have updated the `.gmo` file to add spawn points, we could reload it by selecting Module > Reload current module from the main menu, but we also want to see what happens on startup when the `initiative_seed` setting is in place, so let's quit and restart GMA. After it restarts, bring up the initiative tracker. First of all, we see that the PCs are already there, courtesy of the `initiative_seed` setting we placed in the configuration file.



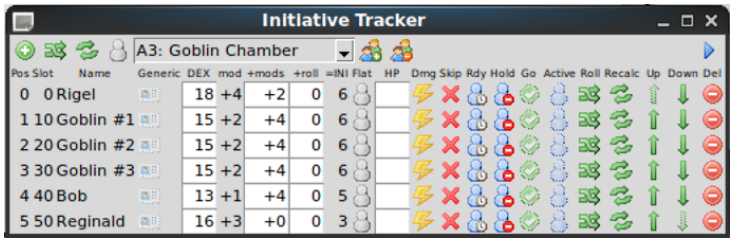
Also note that there is a selection box to select the encounter area at the top of the tracker. This allows us to choose an encounter area to set up for combat with a single click.

Select the encounter area and click the button.





This adds the monsters in that area to the initiative list and creates health trackers for them, so all we have to do is add their creature tokens to the mapper and we're ready to go with combat.



If you hold the shift key (⇧) while clicking on the button, then the creature will *not* be added to the initiative list nor spawned to the map. Instead, GMA will merely update the GM's die roller presets to include the die rolls for the monster(s) in that encounter area. (This is useful if, for instance, you remove the die-roll presets accidentally and want them restored without disturbing the rest of the combat being tracked.)

If you hold the alt key (Alt or ⌥ on macOS) while clicking the button, then the creature is added to initiative and die-roll presets, but its token is *not* spawned onto the map. (This is useful if you have already placed their token on the map and don't want GMA to move it by respawning it now.)

Ad-Hoc and Staggered Monster Placement

The area definition in the `.gmo` file assumes that all creatures involved in the encounter will appear at once, and in fixed locations on the map. However, in practice this isn't always true.

When specifying the spawn points for creatures using the `spawn` field inside a creature's `[m]` stanza, you have the ability to handle both of these special cases.

If you only want some creatures to appear at the start of combat, place the spawn points for the remaining creatures in parentheses.

For example, given the monster stanza

```
[m b1_goblins]
species=Goblin
random_qty=3
spawn=F8 (F9) (F9)
```

only the first goblin will be spawned onto the map, at map coordinates F8. This will also be the only goblin (called "Goblin #1" by default) to appear in the initiative list that the players will see, although all three will be on the GM's initiative tracker and will have had their initiative rolled along with the others.

Goblin #2 and Goblin #3 will show on the initiative tracker as if the GM had pressed the "Skip" button (✖), effectively keeping them out of combat for the time being. When the GM is ready for Goblin #2 to enter the fray, she merely presses the "Active" button (👤) to place them on active status. This makes their position in the initiative list visible to the players and will then place the creature's token at map coordinates F9. Since Goblin #3's spawn coordinates are also listed in the file as F9, we assume the previous one will move out of the way before that one is added to the combat.

Although Goblin #2 and Goblin #3 will not appear on any of the players' maps, they will appear on the GM's map in the "hidden" state (shown visually with a semi-transparent creature token if such an image was prepared for it as described on pp. 81 and 535).

What happens if you don't know in advance where you will have the creature appear on the map? Simply enter a question mark ('?') in place of the map coordinates. When it is time to place the creature on the map, GMA will prompt the GM to enter the map coordinates manually. You can combine both of these notations, so the statement

```
spawn=F8 ? (F9) (?)
```

means that, of four creatures involved in the encounter, the first two will be placed on the map as soon as the encounter is loaded. The first will appear at F8, while the second will appear wherever the GM manually directs GMA to place it. The other two will be added later when the GM puts them on Active status. One will appear at F9, and the other will appear where the GM manually directs.

Entering and Exiting Combat

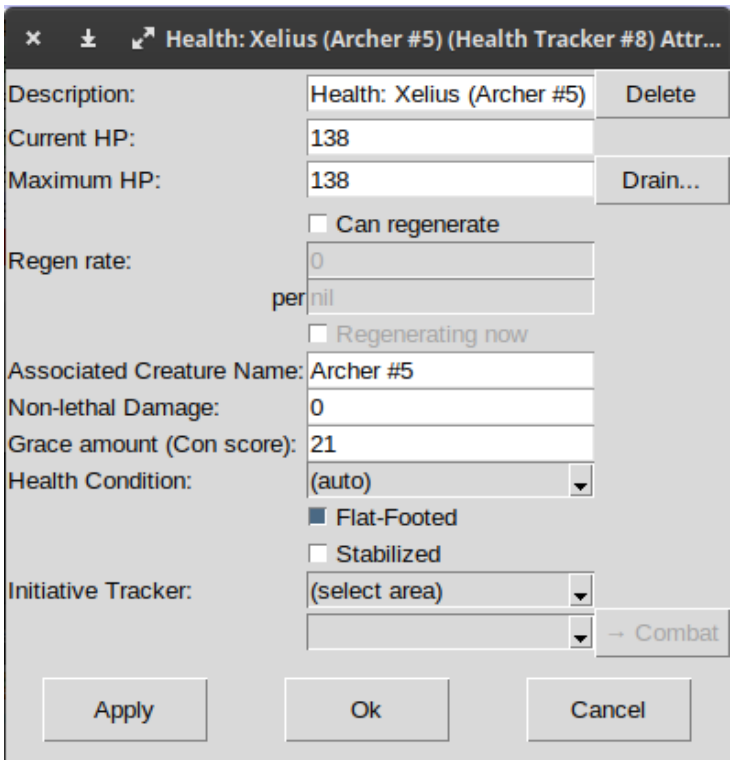
Of course, it's not always the case that a monster will be encountered and dispatched in a single place and time. It may run away, only to be encountered somewhere else. We need to handle this in a way that will keep tracking its individual heath stats so that when it appears in another combat encounter we still have an accurate description of it.

If a creature is defeated (i.e., killed) in combat, the standard procedure is to press the "Del" button (⛔) in the initiative tracker. This removes the creature from the initiave tracker itself and also removes its health tracker from the game clock. The creature is gone from GMA's sense of reality. You

could bring it back by manually entering it, or you could bring back the entire encounter in which it appears.

However, if the creature is leaving combat but is still an active part of the world and will be seen again, a better approach would be to shift-click on the "Del" button, which removes it from the initiative tracker but leaves its health tracker behind. (This will also be the case if GMA is restarted before the creature is removed from initiative—the initiative tracker won't survive the GMA program's restart but the health tracker will.)

Now when the creature is to be re-introduced into combat, click on its health tracker to bring up the health tracker editor window.



At the bottom of this editor window you will find a selector to locate the stanza of the `.gmo` file which describes this monster. Select the encounter area in the top list, then the specific monster instance the the lower one. Then press the →Combat button.



This will insert that specific monster into the initiative tracker, keeping its current health stats as they have been tracked to this point. You will be prompted to say where you'd like the map token to be spawned, since the creature most likely won't still be in the location indicated in the `.gmo` file.

Extra Non-Combatant Creatures

Sometimes you want to include creature tokens on the map which you don't intend to have participating in the combat, but need to note their location on the battle grid. These are like extras on a movie set, so the main characters have a crowd of background characters to fill in the scene. You can also use this to populate livestock in a nearby pen.

To arrange for this, add an `extra_monsters` field in the `area` stanza of your `.gmo` file:

```
[area A3]
title=Goblin Compound
monsters=guards
extra_monsters=Cow|L|black|B17,B20 Pig|M|black|C19
```

Now, when area A3 is loaded on the initiative tracker, the monster(s) defined in the `[m guards]` stanza are loaded and tracked as usual, but in addition, creature tokens for two cows and a pig are placed on the map at coordinates B17, B20, and C19 respectively, but no health status or initiative tracking will be done for the cows and pig at all.

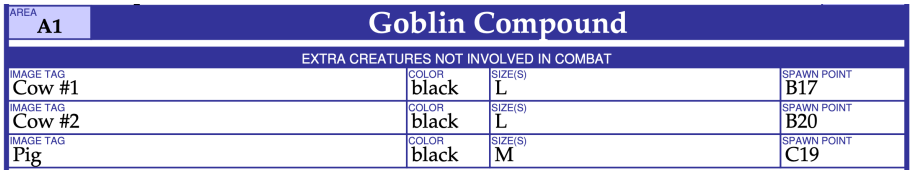
If for some reason the creatures listed in `extra_monsters` happen to enter combat, you will need to manually add them to the initiative tracker at that time.

Each extra monster is separated by the others in the `extra_monsters` field by spaces and/or newlines. Each entry in this list has four bar-separated fields which indicate the creature's `image_tag`, size(s), threat zone/nameplate color, and spawnpoint(s). If more than one spawnpoint is listed, that many copies of the creature will be placed on the map. If multiple sizes are given, as in the `poly_sizes` field, that indicates a disguised or shapeshifting creature.

You can indicate that you want the creature to initially be placed on the map in hidden state by enclosing the spawnpoint in parentheses, such as "`(A12)`" or "`(?)`".

See `gma-monster-sheet`(5) starting on page 433 in Appendix C for the full details.

When printed out, the encounter sheet will show the list of extra monsters at the top of the area, like so:



AREA **A1** — **Goblin Compound**

EXTRA CREATURES NOT INVOLVED IN COMBAT

IMAGE TAG	COLOR	SIZE(S)	SPAWN POINT
Cow #1	black	L	B17
Cow #2	black	L	B20
Pig	black	M	C19

Controlling Printed Output

If you'll be printing out your encounter sheets you may find that you are using a lot of paper to repeat the same monster descriptions over and over (if, say, you have a large number of goblins which are encountered in many separate areas).

One strategy to address this is to have the first occurrence of the monster include all information, but only show an abbreviated version for the subsequent entries. This is controlled by adding a `verbosity` field to the monster entry, like this:

```
[m b1_goblins]
species=Goblin
random_qty=3
verbosity=0
```

The value for this field is an integer in the range 0–4, which controls how much information will be printed.

`verbosity=0` Print the very least amount of information. Essentially this will include the first line of the monster entry, template names, and the individual monsters' hit points and (if applicable) spell names. The latter is provided so you can check off the boxes to track hit points and spell use.

`verbosity=1` Print the full monster description, excluding spell information.

`verbosity=2` Print the full monster description, including the names and basic stats of spells, but without any spell description text.

`verbosity=3` As above, but include a compressed form of the spell description text.

`verbosity=4` Print the complete monster entry. This is the default.

CHAPTER 8

Tracking Progress

I like progress but I hate change.
—Jon Bon Jovi

In chapter 3, we introduced the concept of counters (starting on p. 28). We saw how these are used to keep track of the amount of some sort of *thing* in play: hit points, psionic power points, the number of marbles in a bowl for some unexplained reason, but always a quantity of some kind of object or subtance, whether it's tangible or conceptual—*nouns*, if you will.

And now, for something completely different... using counters to keep track of *verbs*.[1] Which is to say, to track actions instead of things.

Sometimes you also need to track the progress being made through the steps of some sort of process or procedure. This might be the steps of assembly for some apparatus, following a recipe or formula to create a potion, performing a long multi-stage ritual, or even the GM keeping track of how a long-running plot is moving along in the background.

Counters can serve this purpose as well, if we just think of the "number" they are "counting" as being the *step number* in the overall process. If, for example, we are tracking a five-step process, we would set up a counter with a maximum value of 5. If the initial value of the counter is 0, then we haven't started the process yet. When step 1 of the process is completed, we move the counter to value 1, and so on, until the counter reaches 5, meaning that all five out of five steps have been done and the process is complete.

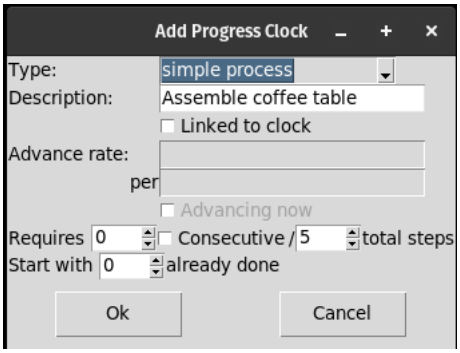
[1] Apologies to Monty Python.

We *could* simply do that, of course, but the astute reader has already guessed that if that were all there was to the story we wouldn't have started an entire new chapter just to say that. Yes, you're very clever. Just keep reading.

To track things like this more easily, we will use a specialized kind of a counter called a *progress clock*. This is already set up to assume the meaning for its current and maximum values as described above, plus it adds controls and behaviors that make it easier to use for tracking the kinds of processes and events that come up in fantasy role-playing games.

Visually, progress clocks are represented as a circle divided up into segments or "pie slices" for each of the steps in the process being tracked.[2] As each step is completed, you "fill in" the corresponding "slice" of the "pie chart" until the whole circle is solid.

In GMA, creating a new progress clock counter is started by clicking the button on the toolbar or selecting Clock > Add Progress Clock... from the menu.



You may recognize this as a modified version of the counter creation dialog we saw in Chapter 3. The fields are:

Type: Select the type of progress clock from the pre-defined list here. By default, GMA comes with "simple process" defined, which should suffice for tracking simple processes, as the name suggests. We will see below how to define more kinds of clocks to do fancier things.

Description: This is a brief description of the process we will use so we know what it is we're tracking.

Requires: If "completion" of the process only requires some of the steps to be done, indicate that number here. Leaving this at 0 means all steps need to be done for the process to be considered complete.

Consecutive: If this box is checked, then the required number of steps need to be *consecutive* in order to count toward successful completion.

[2] As far as the author is aware, progress clocks as a game mechanic appeared first in Apocalypse World and seems to largely have become popular thanks to Blades in the Dark, but is now a popular tool that shows up across the TTFRPG landscape.

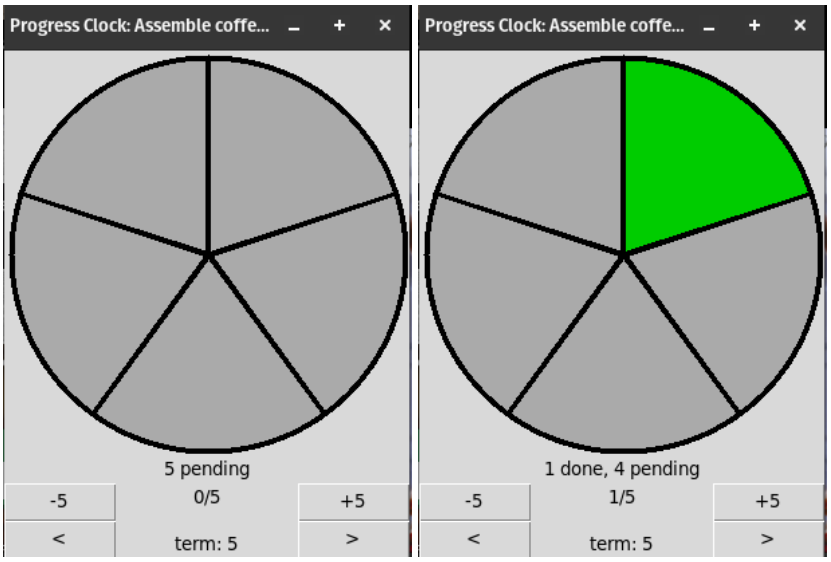


Figure 8.1: Example progress clock (no steps / one step completed)

That's not going to make sense until we get to some more complex examples later, so don't worry about that for now.

total steps: The total number of steps in the process.

Start with ... already done: Begin the progress clock with a number of steps already marked as done.

Here we specified a five-step process to assemble a coffee table. If we click the Ok button, a new progress clock event is added to our event list:

We also get a new window with the progress clock display (Figure 8.1).

Here we see the five segments in the progress clock along with control buttons to advance (>) the process by marking a step as completed, back up a step (<), advance five steps (+5), and back up five steps (−5).

If we click on the > button once, we see now that we're one step into the process (Figure 8.1). This is also summarized at the bottom of the window as it says there is "1 done, 4 pending" and the progress of "1/5" of the steps qualify toward the completion criteria. The "term: 5" at the bottom means that the termination condition is five steps completed.

Normal counters also have a "regeneration" feature where they may automatically recover points at a given rate as the game clock advances. Since progress clocks are a kind of counter, they also have this capability. Looking back at the creation dialog above, this is enabled by checking the "Linked to clock" box and filling in the following fields:

Linked to clock: If checked, the progress clock automatically advances (i.e., marks steps as completed) as the game clock advances.

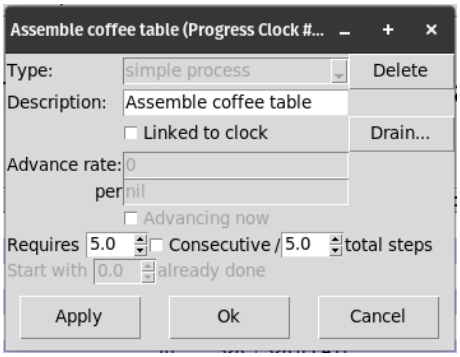
Advance rate: How many steps to advance at a time.

per: The time interval between each such advance.

Advancing now: If checked, the feature is enabled. Otherwise, although the clock may be advancing, the progress clock is ignoring the passage of time until the box is checked again. This control is also available by clicking on the **State** column of the GMA event list.

If we close the progress clock window, we can get it back again (or, if needed, get another window showing the same clock) by clicking the mouse on the progress clock line in the GMA event list in the **ID** or **Value** column.

As with all counters, clicking in the **Attributes** column brings back the editor dialog to allow us to change the parameters of the counter again. Note, however, that once the progress clock has been created, you cannot change its type or number of steps to mark as initially done, since that ship has already sailed.



Now that we're editing an existing progress clock, there are two additional buttons on the dialog: clicking on Delete removes this progress clock from the game. Clicking on Drain... creates a drain attached to the progress clock, just like a normal counter—it will remove progress from the process tracker at a set schedule.

Failure is an Option

Some processes are more complex, like Pathfinder rituals. These are group activities which take a number of minutes to perform, during which there are stages where someone needs to make a skill check, the outcome of which may be a success or a failure. In order for the ritual itself to succeed, at least half of the steps must have succeeded.

We can model the terminal condition for success easily enough by specifying that half the steps are required, but we don't have a way to indicate whether a step was a success or failure. Up until this point, we could only

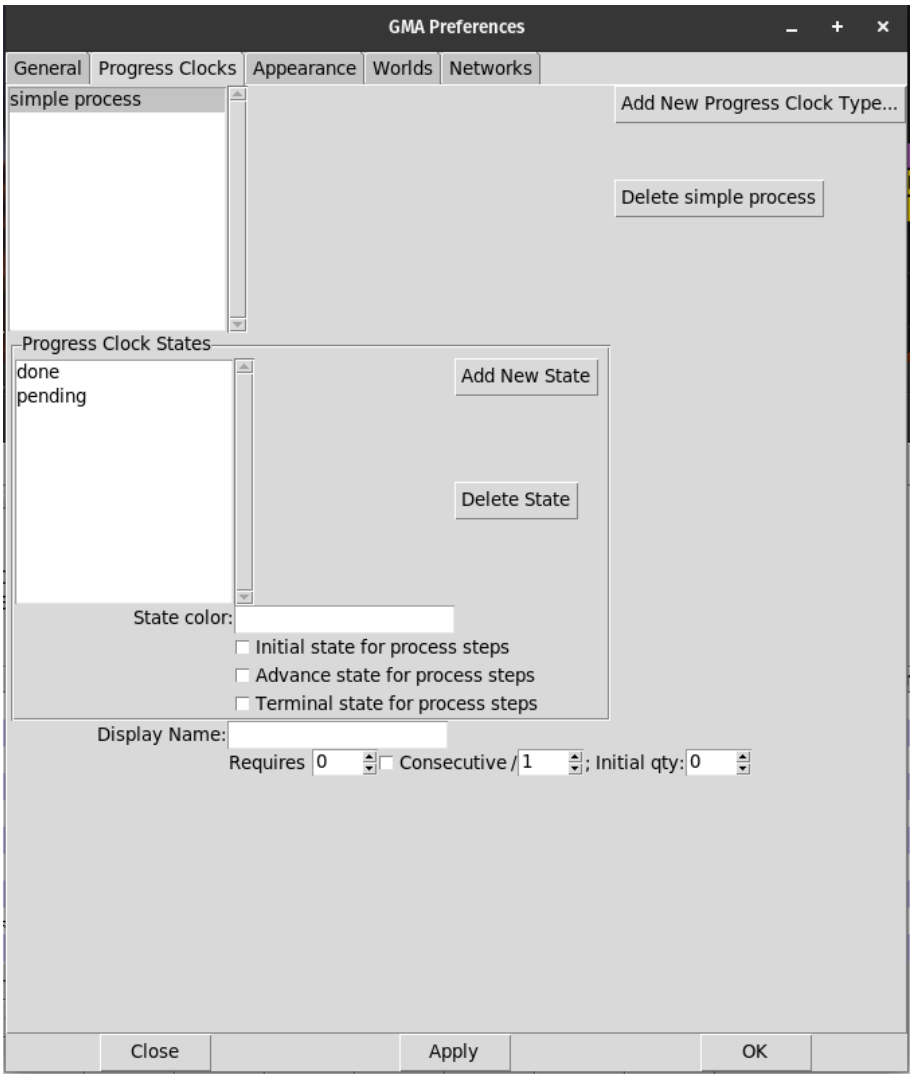


Figure 8.2: Progress Clock Type Editor

mark a step as “pending” or “done.” Now we need three different states a step can be in—“pending,” “successful,” and “failed.”

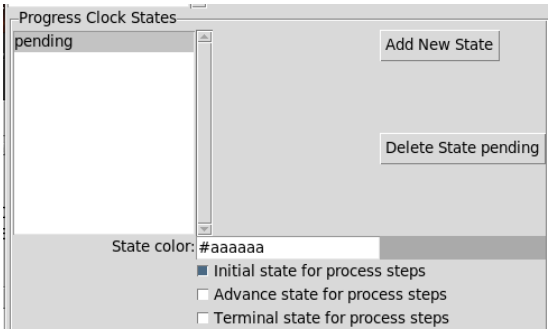
In order to do this, let’s define a new type of progress clock.

Select Edit » Preferences... from the menu, and click on Progress Clocks. You will see the progress clock type editor shown in Figure 8.2. Start a new one by clicking Add New Progress Clock Type.... When prompted, give it a name like “ritual”.

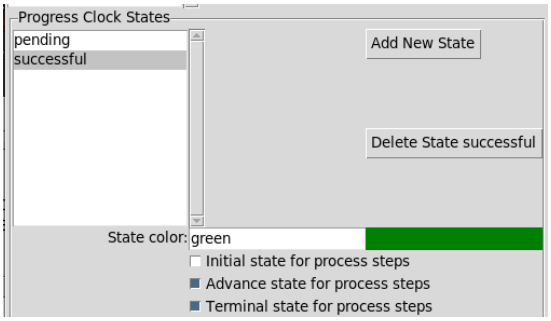
Click on `ritual` in the list of types at the top of the window. In the

Progress Clock States box below it, you'll see that the two default states, `done` and `pending`, were created for us. We don't need the `done` one so let's delete it. Click on `done` to select it and click on the Delete State done button.

Click on `pending`. You should see that it has the "Initial state" box checked as shown. It should also be set by default to a neutral gray color. You can change that if you want to or leave it as it is. This will be the initial state that ritual process steps start out with before they are completed.

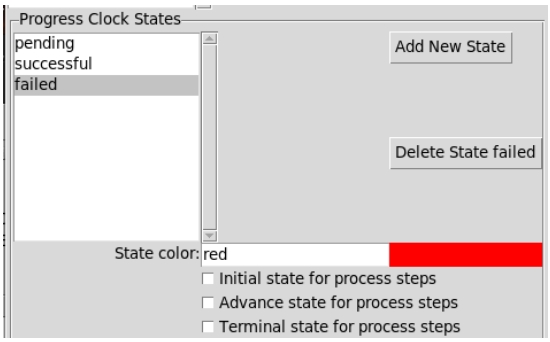


Add a new state by clicking the Add New State button. When prompted, call it "successful". Click on `successful` in the list. Set the color as desired and make sure the "Terminal state" and "Advance state" boxes are checked.



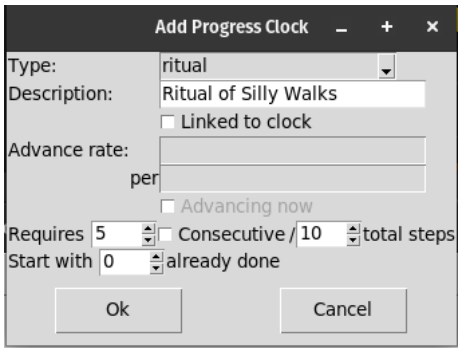
By checking the "Terminal state" box, you indicate that a step marked as "successful" counts toward completing the ritual. By checking the "Advance state" box, you indicate that when you click the advance controls on the progress clock to mark progress, it will mark the next step as "successful" instead of some other state.

Add one more new state by clicking the Add New State button and calling this one "failed". Click on `failed` in the list. Set its color and leave all the boxes unchecked.



Click OK.

Now we can create a new progress clock and select "ritual" as its type, 10 total steps, 5 of which are required for success.



Once we commit this to the tracker, we can start tracking our way through the ritual. Each time a successful skill check is made, we can simply advance the clock one step because we set "successful" as the advance step for the ritual-type progress clock (Figure 8.3).

So far, so good, but let's say on the fourth step the skill check failed. Now we need to indicate that. To set an arbitrary state on any step in the process, simply click on that step directly on the clock. Every time you click on it, the step "pie slice" will cycle through the possible states. We'll click on the next one until it shows failed. Then, as we click advance again, we see that although five steps have been completed, the overall progress still shows "4/10" since only four of them count toward the terminal condition since one of them failed. (Figure 8.4.) Say the next two failed and then another success. At this point, we have satisfied the terminal condition for the ritual so the clock shows that it has been completed. We can continue to run out the clock, marking the remaining two steps as either successes or failures, but in any case we have met the required conditions for success of the overall process.

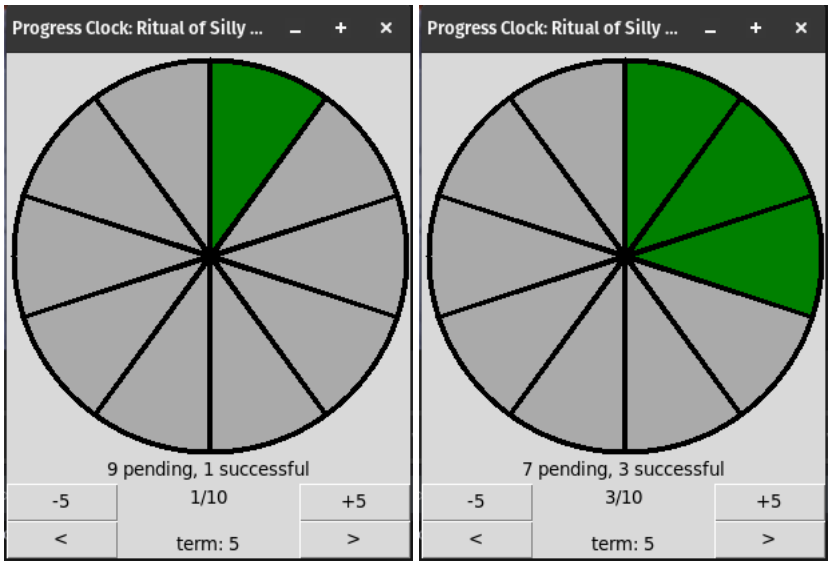


Figure 8.3: Ritual with Some Successes

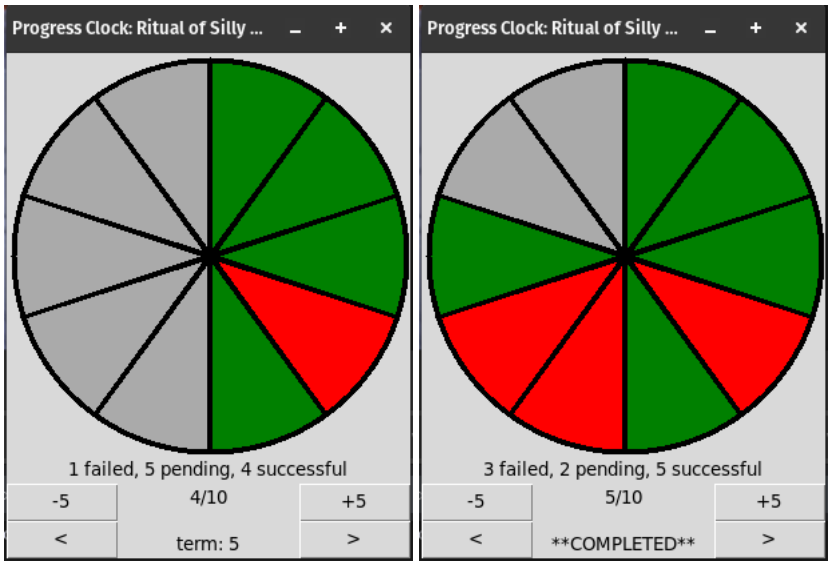


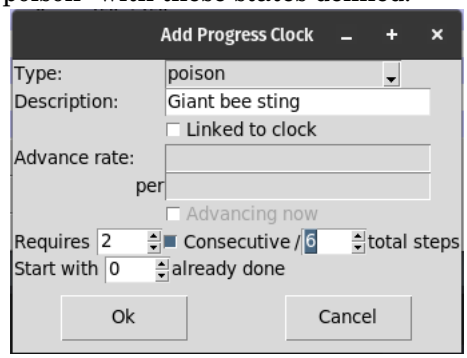
Figure 8.4: Ritual with Some Successes and Failures

Consecutive Successes

Sometimes the success criteria is a little more complicated than what we showed for rituals. In Pathfinder, for example, some poisons require multiple saving throws to survive their effects, and the more potent of these require them to be *consecutive*.

Let's say someone gets poisoned with a toxin that causes damage to their system over the course of 6 rounds. Each round they can attempt a saving throw to withstand the effects that round. If they get two of those saving throws *in a row*, however, the poison has run its course early and has no further effect on them.

To model this, we need a progress clock type with a "pending," "successful," and "failed" state like we made for rituals, since we need to know at each step if we made or failed the saving throw. Let's say we have a progress clock type called "poison" with these states defined.



When we create this progress clock, we'll say that we require *2 consecutive* successes out of a total of 6 steps. You can see as we record successes and failures how the clock requires the successes to be consecutive to count (Figure 8.5).

Pools of Resources

You can also use progress clocks as a way to visualize things we normally use the regular counters for too, like pools of *ki* points. For this, you may find it useful to fill in a value in the "Start with… already done" field of the Add Progress Clock dialog so that the progress clock is created with a number of steps already marked as done. Then you can advance or rewind the clock as needed to track the value of the resource you're tracking using the clock.

Progress Clocks for Monsters

In Chapter 7, we saw how to set up encounters to automate much of running combat. In the encounter (`.gmo`) file, you can list progress clocks that you

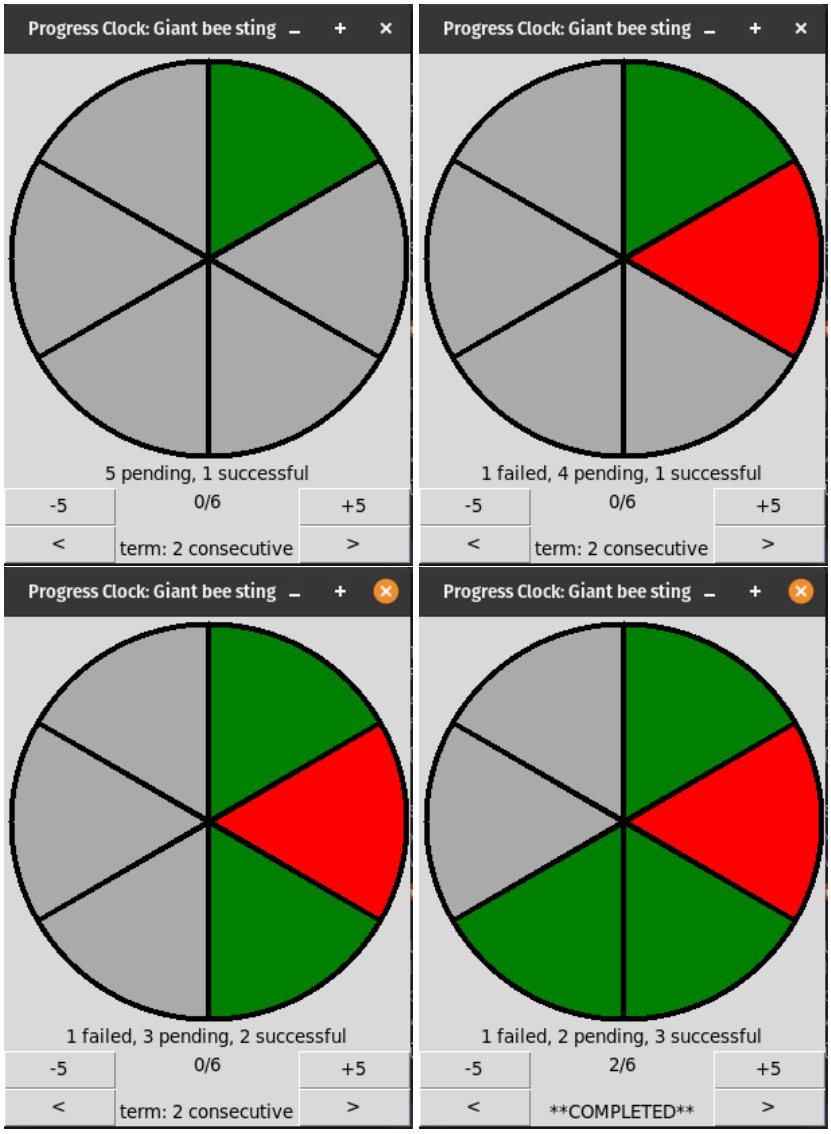


Figure 8.5: Requiring Consecutive Saving Throws

need when running a monster in combat by including a `progress` field for that creature (see p. 407 for full details).

For example, if you include this:

```
progress:
     *plot=5|simple process
     rites=5/10|ritual
     special=@42/100||empty=black,chaos=red,whimsy=green,off=ye
llow,full=blue
```

This defines three progress clocks for that monster. The first one, because it begins with an asterisk, will be created immediately when the creature is added to the initiative tracker. It is named "plot", is of the type "simple process," and has 5 steps. The other two, "ritual" and "special," appear in a drop-down list for the monster in the initiative tracker so you can activate them as needed during the encounter. When you do, the progress clock will be created.

The "rites" clock is of the "ritual" type (which must have been created in your preferences), and requires 5 steps out of 10 total to be successful.

The last one is called "special" and does not have a listed type. Instead, we define the type as a custom type right here in the monster definition by giving a set of ⟨*name*⟩=⟨*color*⟩ pairs separated by commas. The first of these will be the initial state that the process steps start out in, the next will be the default state that the clock advances steps into, and the last one ("full") is the terminal state that indicates that the step counts toward the terminal condition of the process. In this case there will be 100 total steps on the clock, and the clock will begin with 42 of them already marked as completed ("full" state).

In the `.gmo` file, you can also include a field called `imposed_conditions` which lists temporary conditions the creature can impose on other creatures, such as stunning or blinding them (see p. 404).

For example, if a creature has an ability to stun a character for 1d4 rounds and another that poisons them, you might add this field:

```
imposed_conditions:
     shout|stunned||[1d4] rounds
     venom|poisoned||
```

These add items to the monster's initiative entry so you can activate them when needed, and they'll add the condition markers to the afflicted targets' map tokens as well as creating timers to automatically remove them if applicable.

If any of these also need progress clocks, you can specify that as well. For example, if the venom requires 2 consecutive saves out of 5 total and you

want to track that automatically every time the monster stings someone, just change the `imposed_conditions` field to read:

```
imposed_conditions:
     shout|stunned||[1d4] rounds
     venom|poisoned|||poison:2c/5
```

Random Parameters for Progress Clocks

Sometimes the number of steps, initial steps to be completed, and/or the number required are determined randomly rather than being a fixed quantity. For example, we could use a progress clock to keep track of the number of rounds remaining until a dragon's breath has recharged and is ready to be used again.

If a dragon can use this special attack once every 1d4 rounds, that means we effectively need to generate a new progress clock every time, randomly picking a number from 1–4 each time for the number of rounds to track. In the `.gmo` file, instead of using a fixed integer value for the number of steps, required steps for completion, or number of initial steps to start off with as pre-completed, you can put a die-roll expression in square brackets (just like you can do in other places in the `.gmo` file), so our progress clock for the dragon breath could be set up like this:

```
progress:
     *>breath weapon recharge=[1d4]
```

Here we've given it a title ("breath weapon recharge"), specified that it has 1d4 total steps, all of which are required for the process to be considered complete, and left everything else to the defaults (so we will have a standard simple process clock). The "*" at the start is a flag that means to create the progress clock immediately when the dragon enters the arena, and the ">" flag means to connect the progress clock to the game clock so it automatically advances each round.

When the progress clock is created, GMA will roll the 1d4 automatcially, and the result will be locked in as the number of steps for this progress clock from then on, no matter how many times we examine, load, reload, start, stop, or reboot the program. But when the dragon uses its breath weapon, we need to reroll that and get a new progress clock with a new number of steps. To accomplish this, click on the  button in the upper-left corner of the progress clock window. This will destroy the progress clock and generate a new one to take its place, re-rolling any random elements that were used to create it.

CHAPTER 9

BONUS FEAT: CONNECTING GMA AND THE MAPPER

> The essence of a role-playing game is that it is a group, cooperative experience.
>
> —Gary Gygax

UP TO THIS POINT we have put into place some handy automation in the GMA tool, but it would be even better if the GMA program and the mapper could talk to each other, so creature tokens appear automatically on the map for encounters, health and initiative turn order shows in the mapper, etc.

Coincidentally, that's exactly what this chapter is about. To do this, we need to set up a server that will facilitate the communication between these two programs. For now, let's just set up a simple server just for you, the GM, to use on your own computer. We will assume that the server doesn't have access that allows outside users to connect to it (you'll want to make sure that is the case with your local computer security controls).

Go-GMA

The server is part of the Go-GMA package, available from the github repository MadScienceZone/go-gma. If you clone that repository and then type `make` in its top-level directory, it will compile the Go-GMA commmands (including the server) on your machine.

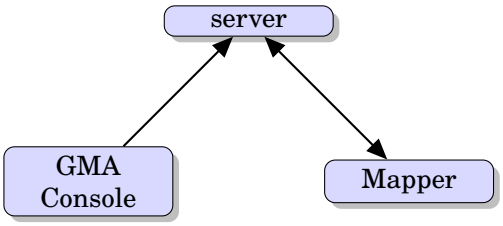


Figure 9.1: Simple GMA Server Configuration

If your `go-gma` directory wasn't cloned into the same parent directory as `gma` was, you'll need to set the environment variable `$GMA_GO_GMA` to indicate where you installed it. Make sure you don't also have `$GMA_IGNORE_GO_GMA` set in your environment.

With that done, you're ready to run the server as outlined below.

Server Basics

The mapper server exists to allow mapper clients to talk to each other (which we'll discuss more in Chapter 10), as well as for other programs like the GMA GM console to connect and control the mapper. While you have a server running, you can have these programs connect to it, and talk to each other through this server.

The server program is run as "`gma go server`".[1] (Don't run it yet—we still need to see how to configure it.) If you're just using it to connect your own mapper client and GM console, you can start the server locally, use it, and then shut it down again when you're finished.

You can visualize the connections as illustrated in Figure 9.1. Communication takes place over TCP port 2323 if you follow the tutorial as written, but if that port is already in use or you would like to use a different one, just use the desired port number everywhere in the instructions that follow instead of 2323.

Setting up a Simple Server

To launch a server locally for your personal use, you can simply start a separate command window and run the command:

[1] Instead of running `gma go server`, you may also run the server directly from the Go-GMA directories by executing `go-gma/cmd/server/server`.

```
$ gma go server -endpoint :2323 -sqlite gmaserver.db
go-gma-server: ... WARNING: authentication not enabled!
go-gma-server: ... configured to listen on ":2323"
go-gma-server: ... using database "gmaserver.db" to store int
ernal state
go-gma-server: ... Server 5.2.1 started
go-gma-server: ... Implements protocol 402 (library supports
minimum 400, maximum 402)
go-gma-server: ... preamble data manager started
go-gma-server: ... starting messsageID generator at 167890352
2
go-gma-server: ... client list manager started
go-gma-server: ... game state manager started
go-gma-server: ... Listening on :2323
```

The value for the `-sqlite` option is the pathname to use for the database the server will use to keep persistent game information. If this file does not yet exist, `server` will create a new one for you.

While the server is running, your mapper and GM console should connect to TCP port 2323. To configure the console to do this, add an appropriate network profile in the GMA preferences by selecting Edit › Preferences... inside GMA or running the stand-alone `gma preferences` program.

In this case, use `localhost` or `127.0.0.1` for the host name and 2323 for the port.

Now when GMA is started, it will connect to your local running server. If you want to run GMA again later without the server, change the selected network profile to one with no hostname value.

Let's start GMA now.

```
$ gma
Connecting to map service at ('127.0.0.1', 2323)
SocketInterface: Attempting connection (try 1 of 20)
SocketInterface: Initial server negotiation...
SocketInterface: Server uses protocol 402
SocketInterface: Server does not need authentication
SocketInterface: Server sign-on completed.
```

To start the mapper so that it connects to the server for just this session, you can include command-line options to initiate the connection:

```
$ gma mapper -- --host 127.0.0.1 --port 2323
```

Or, to make the mapper always connect to the server, place those options in the mapper's preferences via the Edit › Preferences... menu.

Setting up Authentication

While technically this works, and could be used to have multiple GMA Mapper clients talk to each other by connecting them to the running server, our GM console is going to be limited as to what it can do, since it has no way to identify itself to the server as the privileged GM user.

Let's fix that before going further. Stop the GMA program, the Mapper, and the server. (If you started the server as shown above so it's running in the foreground, hitting ctrl + C will terminate it. If it's running in the background, sending an interrupt signal to its process will also work.)

Create a file to hold user passwords for the server. For the sake of this tutorial, let's call it `gmaserver.sec`. Type two passwords into the file, one per line, like so:[2]

```
cookies
R0cksFall
```

The first password ("`cookies`") is the non-privileged credential that players can use to access the server. The second one ("`R0cksFall`") is the privileged one for the GM to use.

Restart the server with the `-password-file` option.

```
$ gma go server -endpoint :2323 -sqlite gmaserver.db \
-password-file gmaserver.sec
go-gma-server: ... authentication enabled via "gmaserver.sec"
go-gma-server: ... configured to listen on ":2323"
go-gma-server: ... using database "gmaserver.db" to store in
ternal state
go-gma-server: ... Server 5.2.1 started
go-gma-server: ... Implements protocol 402 (library supports
minimum 400, maximum 402)
go-gma-server: ... preamble data manager started
go-gma-server: ... client list manager started
go-gma-server: ... game state manager started
go-gma-server: ... starting messsageID generator at 167890459
8
go-gma-server: ... Listening on :2323
```

Configure GMA to use the GM password by adding it to the selected campaign world preferences setting.

Run the `gma` command to get a working GM console with GM privileges on the server.

[2] Of course, you'll want to make up your own passwords, and hopefully ones that are less easily guessed than these examples.

You'll also need to add a password to the Mapper client when connecting:

```
$ gma mapper -- --host 127.0.0.1 --port 2323 \
    --password R0cksFall
```

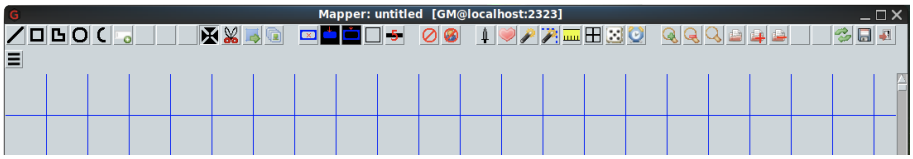
Players should, of course, use the player password:

```
$ gma mapper -- --host 127.0.0.1 --port 2323 \
    --password cookies
```

Really Automating Combat with the GM Console

Now, with a running server, GM console, and mapper client, let's see what we can accomplish.

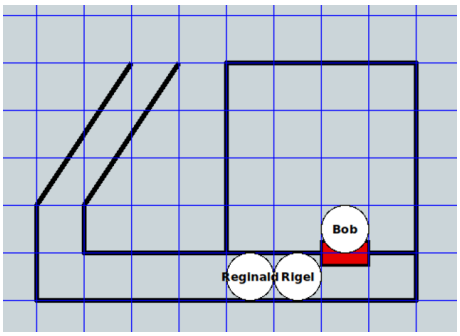
The first thing to notice is that our server connection is noted in the Mapper's window title.



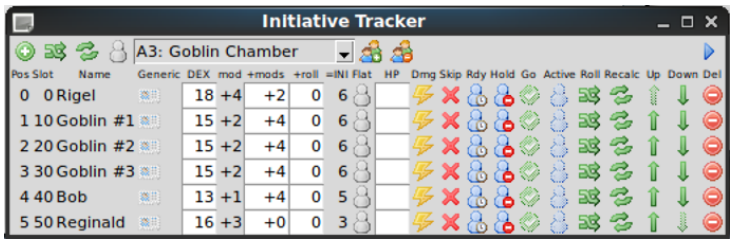
On the GM console, you can control whether Mapper clients display their toolbars by clicking the + Map TB button to turn them on or − Map TB to turn them off. (Clients may be configured to ignore this.)



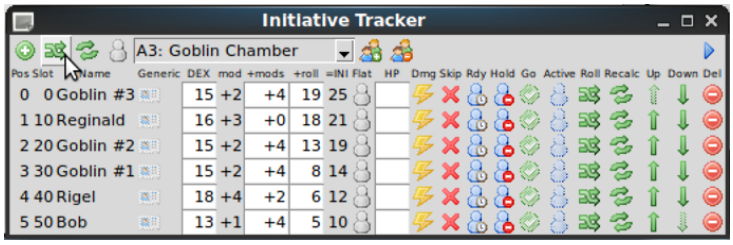
Now let's load the simple map sketch we made earlier and put the PC character tokens on the map. Say Bob just opened the door and entered the room full of goblin guards.



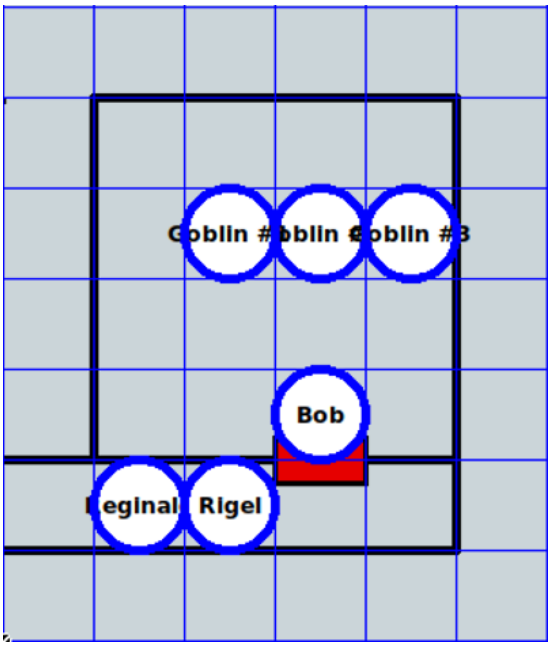
Now it's time for combat. However, since we have prepared the encounter in advance and connected the mapper, we just select the encounter from the initiative tracker's list and click .



Click the roll button to roll initiative for everyone.



In addition to setting up the initiative list and health trackers, we now see that monster tokens have been added to the map for our goblin friends!



The heavy blue borders indicate that they are flat-footed at the start of combat. If necessary, you may need to click the global flat-footed button to set everyone's status at once



(or the individual flat-footed buttons to toggle the states of each participant as appropriate). Now when we put GMA into combat mode, the mapper automatically does so as well, marking the threat zones around each combatant as shown in Figure 9.2.

Unlike what we saw before connecting these programs together, we see that it is only displaying *one* person's threat zone as a fully-shaded area, while the others are just dotted outlines. This indicates that it's Goblin #3's turn to take an action in combat.

We also see health bars displayed under each creature, too, as was described in Appendix I. These are tied directly to the health trackers, so when Goblin #3 moves down and attacks Bob, we see Bob's health tracker display the damage he suffered.

Recall from Chapter 3 that we can set a special timer to indicate that a creature has a condition applied to them, such as being dazzled or unconscious, and that timer will remind us when the condition should be started (in the case of a delayed-onset condition) and/or when it should be removed.

Let's try that again in the connected environment.

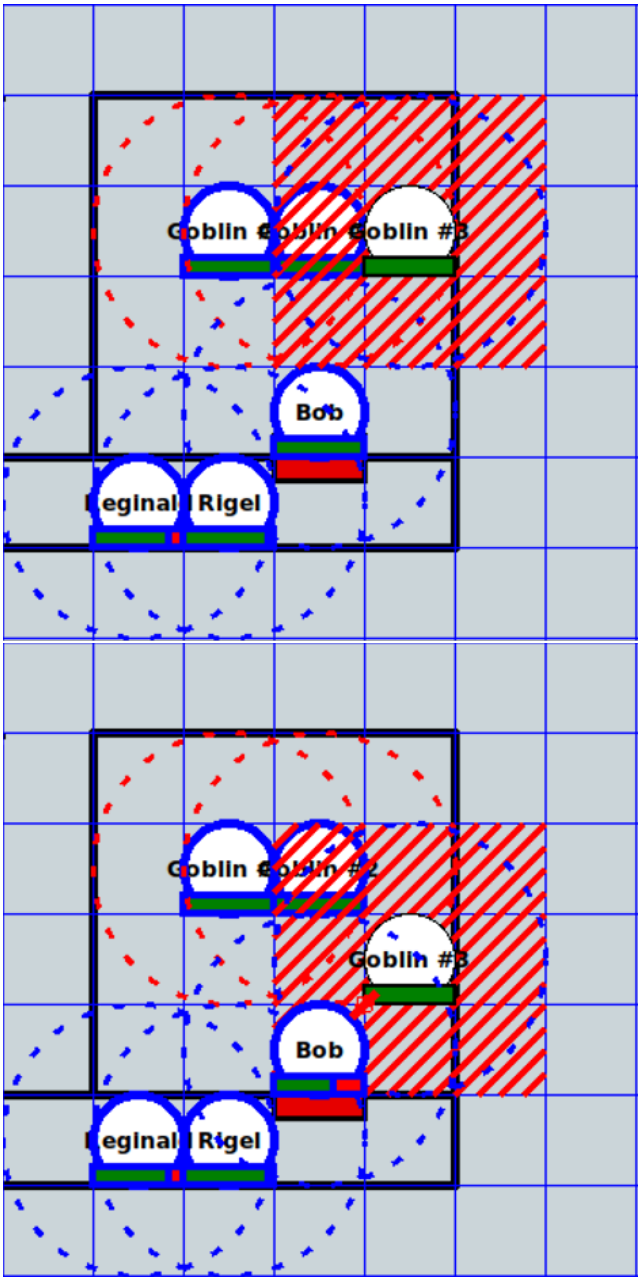
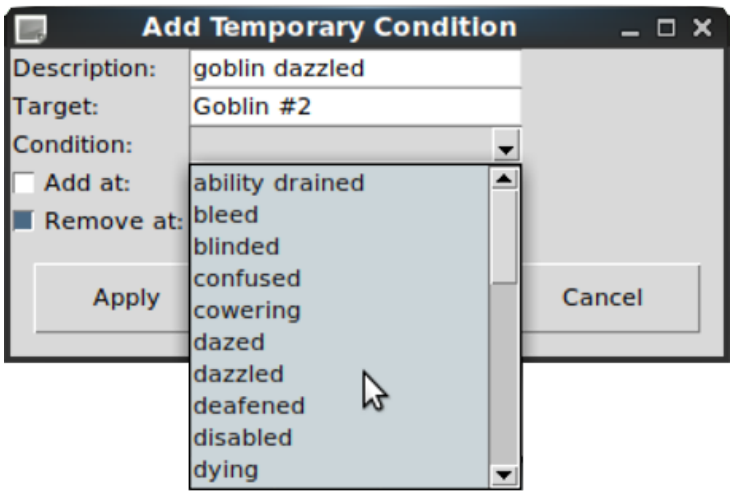


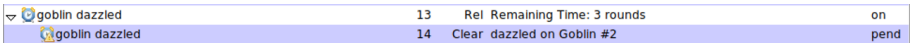
Figure 9.2: Mapper Display of Combat Turns and Health Bars



Now, rather than the name of the condition being a free-form text entry, we are given a list of all of the conditions the server recognizes.[3] Let's choose the *dazzled* condition for Goblin #2, taking effect immediately, lasting for three rounds.

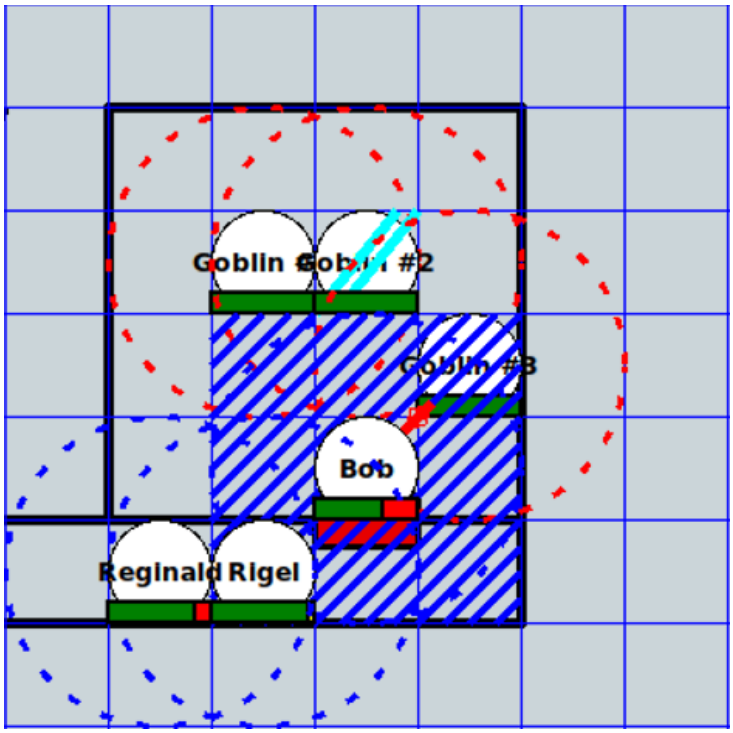


This adds a condition timer to our event list, as we expect:

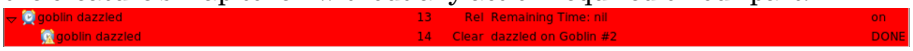


Additionally, we see the goblin's token automatically updated to show their dazzled condition on the mapper:

[3] To change this list, add `DSM` commands to the server's initialization file, as explained in the documentation for `server`(6) on page 597.



Three rounds later, the timer expires, and the marker is removed from the creature's map token without any action required on our part.



Now let's exit out of the mapper (by clicking) and from the GMA program (by choosing File > Quit from the menu).

In the shell window where we started the server, we shut it down by typing ctrl + C .[4]

Setting up the Server for Multi-User Access

As long as we have the server setup, it's more fun if we let everyone in the goup play too. In fact, it's quite useful to let your players run their own mapper clients, all connected to the same server as illustrated in Figure 9.3.

Obviously, in order for this to work you'll need to run the server on a system that allows access from outside systems so your players can connect to it. Appropriate security measures should of course be taken on such a system, which are beyond the scope of this manual to provide guidance.

Earlier in this chapter we created a simple credential file `gmaserver.sec` which holds passwords for players and the GM to use. This is essential for

[4] If the server is running in the background, just send it an INT signal to gracefully shut it down.

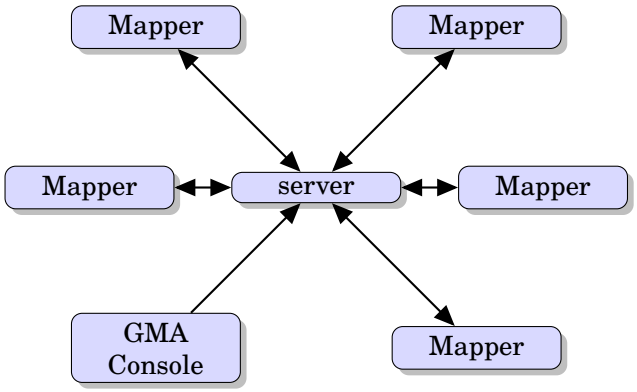


Figure 9.3: Multi-Player GMA Server Configuration

taking the server to the multi-user level, so that things that are for GM use only stay in the hands of the GM.

Tell your players to include the `--password` option when connecting their mappers, and to use a username to distinguish them (a good option is to use their character name), such as:

```
$ gma mapper -- --host example.com --port 2323 \
    --password cookies --username Bob
```

or to put these bits of information in their mapper preferences settings via the Edit › Preferences... menu. If the `username` option is not specified, then their name shown on the server will be whatever name they logged into their local computer account with.

As GM, your `username` will always be "GM" on the server. You may also wish to add another setting in your campaign world preferences—the "Blur HP" number.

This tells the GM console that when reporting creature health stats to the connected mapper clients, it should "blur" the hit point values. As written here, it will only report them to within 10% of their actual values. This helps avoid too much of a "video game" feel where the players would have too much insight into their opponents' hit point totals. This way they have an approximate feeling of the state of their opponents. Adjust this value to your tastes, or leave it out to have the health bars be completely accurate.

Your users may want to each have their own personal passwords. This is a little more work to manage but has the benefit of making it much more difficult for users to see each other's private chat messages or to impersonate each other on the game server. In this case, simply add additional lines to the `gmaserver.sec` file (or whatever you named it), after the general shared user password and the GM password. Each of these lines consists of

a user name and password separated by a colon. For example:

```
cookies
R0cksFall
Alice:5g/X4mD97
Bob:spaceHamsters!!
```

Now a user may log in with the name "Alice" only if they also give the password "5g/X4mD97", or with the name "Bob" and password "spaceHamsters!!". The GM still logs in with the password "R0cksFall" and any other player name (other than "GM", "Alice", or "Bob") may log in using the password "cookies".

Obviously, you'll want to protect the files which have passwords in them so that other users cannot read them.

To get the most out of our server, we'll want to do a little more setup. We do this by creating a file, say gmaserver.init, with the sign-on commands you want the server to send to anyone connecting to it. For example:

```
// Example Game Server
// Unauthorized access prohibited.
AUTH
AC {"ID": "PC1", "Name": "Bob", "Color": "blue", "Size": "M",
    "CreatureType": 2}
AC {"ID": "PC2", "Name": "Reginald", "Color": "blue",
    "Size": "M", "CreatureType": 2}
AC {"ID": "PC3", "Name": "Rigel", "Color": "blue",
    "Size": "M", "CreatureType": 2}
WORLD {"Calendar": "golarion"}
SYNC
```

You should also indicate the base URL for map files and images which the mapper clients need to retrieve to display them. Do this by adding a ClientSettings value to the WORLD command:

```
WORLD {
   "Calendar": "golarion",
   "ClientSettings": {
      "ImageBaseURL": "https://www.example.com/gma"
   }
}
```

There are many things you can include in this file, such as the definition of custom condition markers for creature tokens and pre-loading PCs to the Mapper's main menu.

See the `server`(6) documentation on page 597 for a thorough explanation of this file's contents.

Once a client connects to a server with these settings, the pop-up context menu that appears when the player right-clicks on the map canvas will include items to place each of these characters on the map in one step, with no need to go through the "Add Player" dialog.

To make the server run these commands upon client connection, we start it with the addition of the `-init-file` option:

```
$ gma go server -endpoint :2323 -sqlite gmaserver.db \
    -password-file gmaserver.sec -init-file gmaserver.init
```

Using the Mapper Console Client

If you want some insight (and control) into the map server's operation, you can attach a console client which just prints out all the messages it sees from the server to your terminal's standard output. See Figure 9.4.

As you can see, the server messages are being interpreted for easier understanding (e.g., the server message "`CO {}`" was printed as "`CombatMode enabled=false`"). There is also a line of blue dots that is slowly growing across the bottom line. Each of those represents a time when the server sent a "ping" packet to make sure our client was still alive and responding. They can generally be ignored.

You can type arbitrary server commands to control things manually, as explained in detail in the `map-console`(6) documentation on page 581.

(There used to be a `map-console` command in the GMA Core package, which was accessed via the command `gma map-console` rather than `gma go map-console`. That version was written in Python and used GMA Core's `SocketInterface` module. However, the newer Go-GMA version of `map-console` is a better implementation of the idea and has now replaced the older script entirely.)

If you have a default server profile saved in your mapper client preferences, the `map-console` client will also use these settings. This allows you to avoid putting your password on the command line where other users on your computer can see it, as well as making it easier to start the client.

Reconnecting to the Server

If a client loses connection to the server, it will try to reconnect automatically. The GMA Mapper usually does this successfully as soon as the connection is lost, but the GM console (`gma` command) may not notice the dropped

```
$ gma go map-console -host example.com -port 2323 \
    -password R0cksFall
GMA mapper console 5.2.1
map-console: ... [client] initial server negotiation...
map-console: ... [client] authenticating to server
map-console: ... [client] authentication sent, awaiting valid
ation.
map-console: ... [client] access granted for GM
map-console: ... [client] listening for server messages to di
spatch...
Connected to server.
map-console: ... Error checking for version updates: The serv
er provided no upgrade information for package "go-gma"
Server protocol 402; using golarion calendar.
Characters Defined:
NAME---- ID- COLOR SIZE
Bob      PC1 blue    M
Reginald PC2 blue    M
Rigel    PC3 blue    M
Condition Codes from Server:
CONDITION------ SHAPE COLOR----- DESCRIPTION-----------------
Available Software Updates:
PACKAGE--- OS-------- ARCH------ VERSION
CombatMode enabled=false
Toolbar enabled=true
AdjustView grid=, xview=0.00%, yview=0.00%
//  no current turn set
//  no current initiative set
//  no current time set
UpdatePeerList
     USERNAME ADDRESS---- CLIENT----------- AU ME PING--
[00]=GM       [::1]:13075 map-console 5.2.1 Y  Y  Active
map-console>
```

Figure 9.4: Example `gma go map-console` output

connection right away. In this case, simply click on the reconnect toolbar icon ( ) or select File > Reconnect to Server from the menu.

Tracking Hidden Creatures

If a creature leaves the field of battle or is moving to a new position out of sight of the players, you can set the creature to "hidden" state on the mapper by right-clicking on the creature, selecting the Toggle Condition for . . . > (hidden) menu item as shown in Figure 9.5.

This will remove the creature from all players' maps, but will leave it on yours, with a semi-transparent image. You can then keep track of its movements until you select the same menu item again to bring it back into normal view.

You can also select multiple creatures by holding the ctrl key while left-clicking each creature, then making the menu selection to change them all in one step.

Shifting Loyalties (or at least colors)

By default, the threat zone for each creature as shown on the map, as well as their nameplate shown above the token image, is drawn in blue for player characters, green for allies, and red for enemies. This color may be set to any color desired by specifying it when manually placing the token on the map, or in a `.gmo` file by placing the color name in a monster's `map_color` field.

But what if you want to change the color of a creature already on the battle grid? This could be the case if, for example, an enemy changes to an ally. To do this, select the set of combatants you want to change color and then press the color icon at the top of the initiative tracker ( ) as shown below.





Figure 9.5: Setting Hidden Condition on a Creature

CHAPTER 10

Diplomacy Check: Chat Channels and Die Rolling

> Random chance plays a huge
> part in everybody's life.
>
> —Gary Gygax

With the multi-user configuration for the server set up as described in Chapter 9, your players have the ability to run their own mapper clients, each of which displays the same information. The mappers are kept in sync with each other via their messages passed back and forth via the server.

If, on Bob's client, he moves his token down the hallway, that token is moved on everyone else's client as well. If Reginald draws a colored rectangle on his client to indicate a new wall he magicked into existence, that rectangle appears on everyone else's clients, too.

There are a couple of additional social features available to everyone now, courtesy of the server connection: the chat channel and on-line die rolls. If playing your game online, where the players are not physically sitting together in the same room, these features combine to make remote game play much more convenient and fair.

Chat Feature in the Mapper Clients

The die roller and chat channel are integrated into the same window, which will automatically open the first time you receive a chat message. You can manually open it by selecting the die icon () in the mapper toolbar.

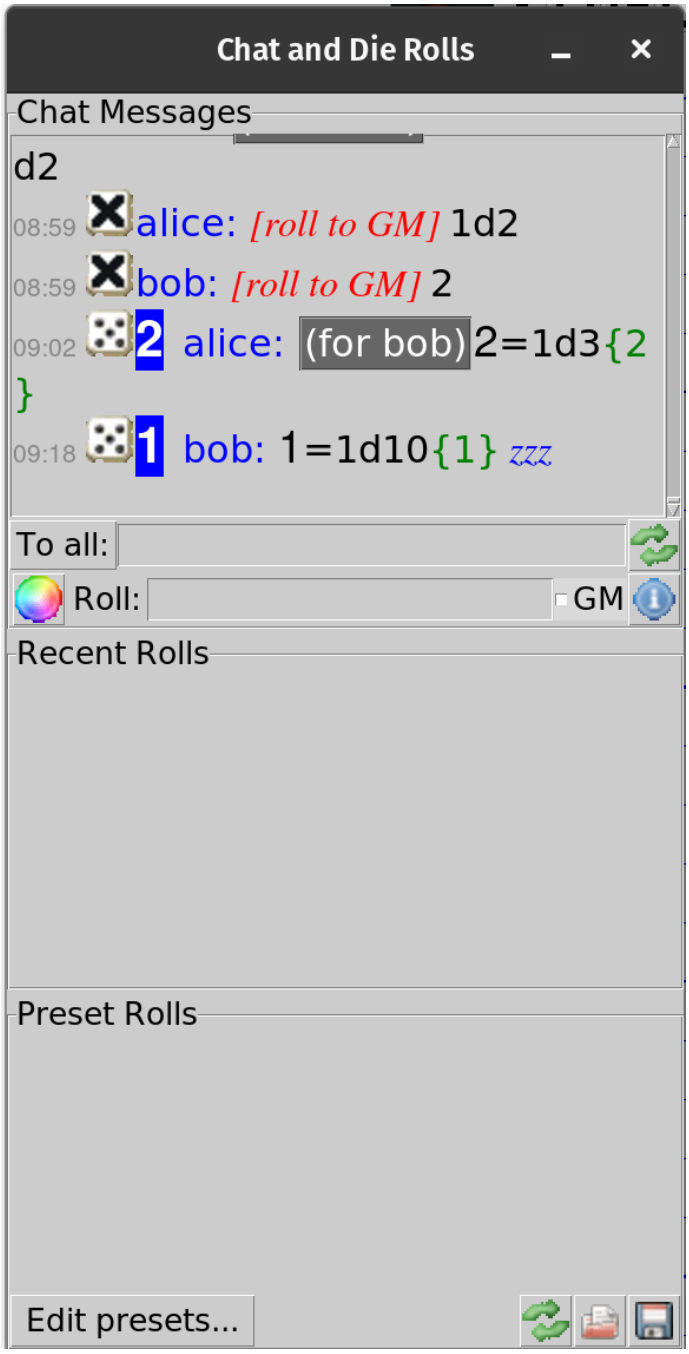


Figure 10.1: Chat/Die Roller Tool Window

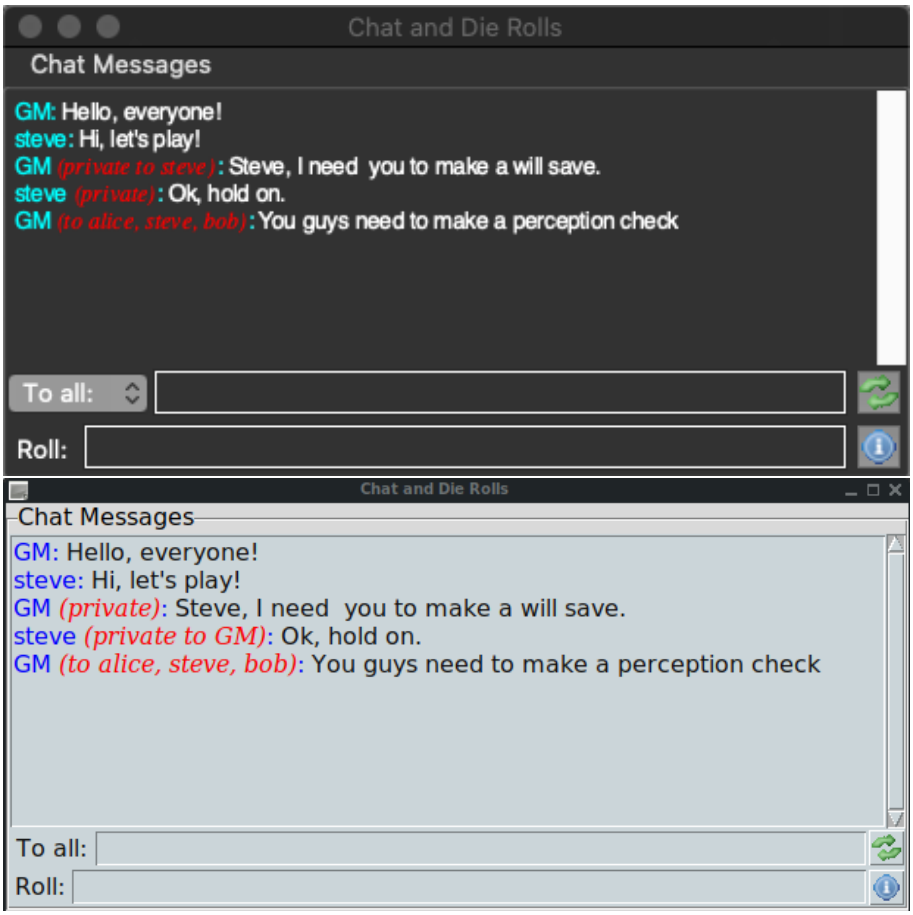


Figure 10.2: Some Chat Messages

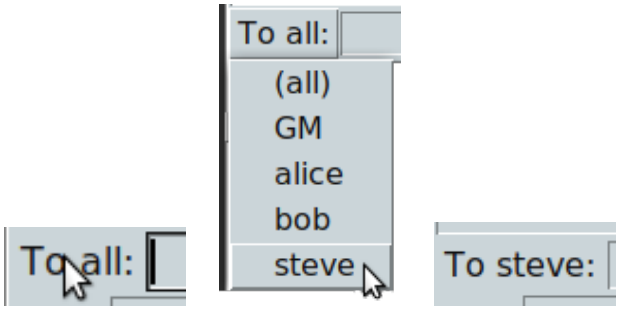
The tool window is divided into three sections, with adjustable "panes" to allow you to control how much space in the window is used for each part. The top area is for the chat tool. Simply type a message into the entry area (just to the right of the To all: button), and press ⏎. Your message will be sent to all connected clients.

Chat messages you receive will appear in this upper window. Broadcast messages will simply have the name of the sender in front of them, while private messages will say so as well. In these examples, the GM's client is shown with a dark background and the player Steve's with a light one, to make them easier to distinguish.

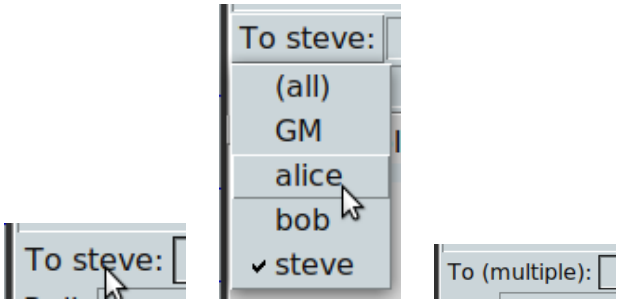
Private and Group Chats

In the example in Figure 10.2, we can see that the GM and Steve each sent a message to the entire channel. Then the GM sent a private message to Steve, to which Steve replied privately back. Then the GM sent a private message only to Alice, Bob, and Steve.

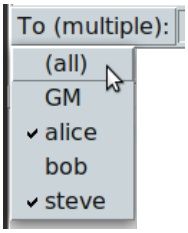
To send a message to a single person, click on the recipient menu button (which starts off saying To all:), select the intended recipient, and then send your private message:



If we click on the menu button again, we can select another recipient. This will add that person to the private conversation.



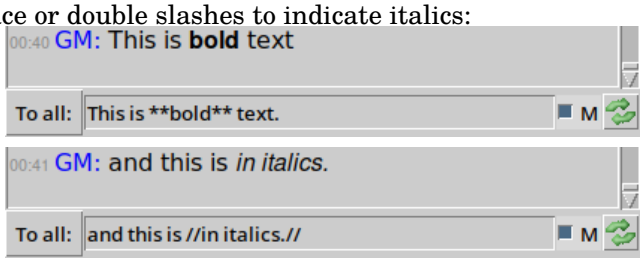
Now we're chatting privately with Alice and Steve. Selecting (all) on the menu clears the private list and returns to sending messages to everyone.



If someone joins or leaves the server, your recipient menu should automatically update so it always contains the current list of connected users. However, if for some reason it gets out of date, you may manually update it by clicking on the reload button (). It is not necessary to have an updated list when sending to all players nor to receive messages from anyone.

Using Markup in Chat Messages

You can use GMA markup formatting codes in your chat messages. These include simple codes such as surrounding text in double asterisks to indicate boldface or double slashes to indicate italics:

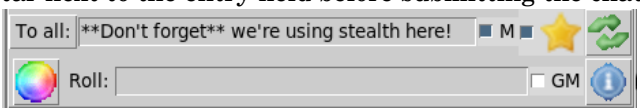


These codes are only recognized if the "M" checkbox to the right of the chat message entry box is selected.

See p. 690 for more information about the formatting codes you can use.

Pinned Chat Messages

If you want to send an important chat message about something that needs to be a long-standing note to the group, such as a list of spells in effect for the encounter, or something important to keep in mind during the session, check the star next to the entry field before submitting the chat message:



This will send it as a "pinned" message which will be displayed in a special area above the other chat messages so it won't scroll out of sight with the rest of the die rolls and incidental chats. The "pinned message" checkbox automatically unchecks after each message to prevent accidentally sending a bunch of messages there, so don't forget to check it again if sending more than one pinned message at a time.



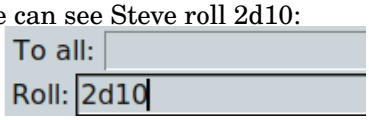
Since these are adjustable panes, you can drag their borders as needed with your mouse to adjust their sizes, so if you don't have any pinned messages you can reduce the size of the top area or even make it disappear, or enlarge it to be big enough to show the messages you need to see.

When you want to clear all the messages out of the pinned chat window, just select Tools » Clear Pinned Chat Messages from the main menu.

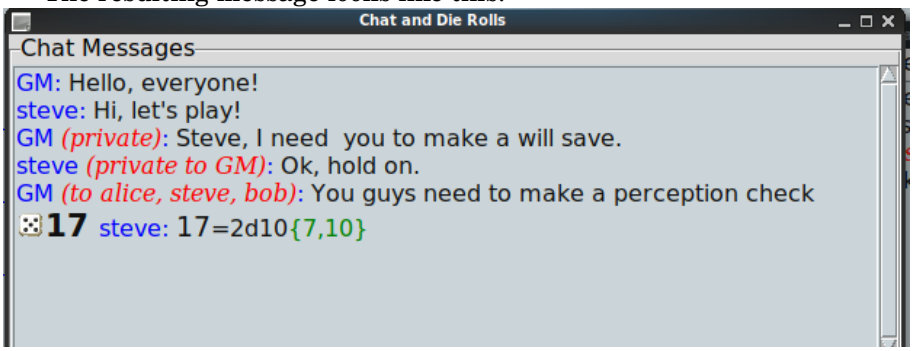
Rolling Dice

Everything from here down will concern die rolls, but the above description applies to them as well, since your die rolls will be sent to the currently-selected chat recipient list, and will be sent over the chat channel just like messages are.

To roll some dice, type the die-roll expression into the "Roll:" line and press the ↵ key. The server will roll your dice and send the results on the chat channel. Here we can see Steve roll 2d10:



The resulting message looks like this:



You can see that the result of the die roll is 17. This is written up front (along with a special die icon to distinguish this from regular chat messages). After the attribution, we see the details of what went into that result. In this case, the two 10-sided dice rolled 7 and 10.

It's that simple. Well, maybe not quite so fast... There are many kinds of die rolls and various modifiers which may come into play, many of which are supported directly by this system. Generally speaking, most of the die roll descriptions you would think of, such as "`d20+5`" or "`3d6+3+1d4 fire`" will work as intended.

More precisely, a die roll expression consists of a set of individual die rolls (like "`2d20`") and/or constants (plain numbers like "3" or "5"), separated from each other by the operators "+", "-", "*", "//", "<=", and ">=", which cause the values around them to be added, subtracted, multiplied, divided, and constrained to a maximum or minimum value, respectively.[1] You may also use "×" and "÷" for multiplication and division. You can also use "≤" and "≥" instead of "<=" and ">=". The standard algebraic order of operations is followed. Parentheses may be used to provide explicit sub-expression grouping. ("<=" and ">=" have the highest precedence other than unary minus, so they will only affect the values immediately around them, but you can use parentheses to control that.)

[1] Why use "//" instead of "/" for division? Reasons. At least 3 reasons, actually, but they're not important to explain at the moment. Just trust us.

Thus, the die roll expression "`d20 - 3 + 3d6 * 2`" will roll a d20, subtract 3 from the result, then roll 3d6 (adding them together), double it, and add that product to the total. If the d20 rolled 17 and the 3d6 rolled 2, 3, and 1, then the result would be $(17-3+((2+3+1)\times 2))=26$.

Fancy Things You Can Do With Each Die Roll or Constants

The following applies to each die roll or constant (between the math operators).

Arbitrary text at the end of a die roll or constant can be used to label what that value means in context, such as "`3d6 falling damage`" or "`d20 + 1 competence + 2 potion`".

The die roll consists of the number of dice (default 1), the letter `d`, and the number of sides on each die ("`d20`", "`3d6`", "`10d4`", or even "`d%`" since "`d%`" may be used as a shorthand for "`d100`"). Fractions are allowed for the number of dice, so saying "`1/2d10`" rolls half of a d10 (actually rolls 1d10 and divides the result by 2).

Placing a "`>`" in front of the die roll causes the first die to be forced to its maximum value. For example, "`>5d8`" rolls 5d8 but the first d8 will always be 8. This is useful for things like hit points.

Following the die roll with "`best of` ⟨*n*⟩" or "`worst of` ⟨*n*⟩" will cause that die roll to be repeated ⟨*n*⟩ times, taking the best (or worst) result. For example, "`d20 best of 2`" will take the better of two attempts at rolling a d20. This applies only to the immediately preceding die roll, so "`2d10 best of 3 + 1d4`" will roll 2d10 three times, picking the best 2d10 result, then roll 1d4, adding that result to the best roll from the previous part.

The `<=` and `>=` operators enforce minimum and maximum values. In general, the expression "⟨*x*⟩`<=`⟨*y*⟩" means "take the value of ⟨*x*⟩ but it must be less than or equal to ⟨*y*⟩, so if it's greater, just use ⟨*y*⟩." The meaning of `>=` is analogous. Thus, if you are caster level `$CL` and need to roll "3d6 + 1 per caster level (max +5)", that could be expressed as "`3d6+$CL<=5`". These may be combined, as in "⟨*x*⟩`>=4<=10`", which means to take the value of ⟨*x*⟩ but it must be a number at least 4 and at most 10.

Global Modifiers

The following applies to the entire die roll expression typed into the tool.

If you put a descriptive title and equals sign in front of the die roll expression, that title is reported out in the results to label what the die roll is for: "`Rocks Fall = 6d6 bludgeoning`".

After the die roll expression you may place a number of global modifiers which are separated from each other and from the expression by vertical bars ("`|`"). These control how the entire expression is evaluated. They may be any of the following:

| `c`**[**⟨***threat***⟩**][**±⟨***bonus***⟩**]** This indicates that the roll may need a critical confirmation roll to follow it. After making the initial roll (of the entire expression), another one will be made if the first resulted in a natural die roll at least ⟨*threat*⟩ (by default, the maximum face of the die). If ⟨*bonus*⟩ is given, that is added or subtracted from the confirmation roll. For example, "`d20+5|c`" makes an attack roll at +5 and automatically rolls again to confirm a critical hit on a natural 20, while "`d20+7|c19+1`" makes an attack roll at +7, rolling to confirm critical hits on a natural 19 or 20, adding a +1 bonus to the confirmation roll.

  This implies the `sf` option automatically.

| `dc` ⟨***n***⟩ This roll is against a known Difficulty Class of ⟨*n*⟩. Along with the results, it will be reported whether this met the `DC`, and by how much of a margin either way.

| `max` ⟨***n***⟩ The final result will be forced to be no more than ⟨*n*⟩.

| `min` ⟨***n***⟩ The final result will be forced to be at least ⟨*n*⟩.

| `repeat` ⟨***n***⟩ The die roll expression is performed ⟨*n*⟩ times, reporting all results.

| `sf` **[**⟨***success***⟩**[/**⟨***fail***⟩**]]** The die roll (which must be a single die only) is considered to be a success if the die naturally rolled its maximum face value, or a failure if it was a natural 1, regardless of other modifiers in play. You may optionally add labels to be printed in case of success and failure.

| `total` ⟨***n***⟩ The die roll expression is performed repeatedly until the cumulative total of all rolls is at least *n*. Each roll along the way is reported (along with the cumulative total) along the way, so you can see how many rolls it took to reach the total.

| `until` ⟨***n***⟩ The die roll expression is performed repeatedly until the result is at least ⟨*n*⟩. Each roll along the way is reported (in case there are consequences for each failure until a successful result is given).

| `maximized` All dice are taken to be their maximum possible value rather than being random. So "`Rocks Fall = 6d6 Bludgeoning | maximized`" will always result in the value 36.

| `!` Alternative form of the `maximized` modifier for legacy reasons.

Percentile Checks

As a special case, rather than the die roll expressions described above, you may give a die roll expression as a number followed by a percent sign, like "`55%`" or "`20%`". This indicates the probability (as a percentage) of something occurring and may take the form

[⟨*title*⟩=] ⟨*chance*⟩% [⟨*success*⟩[/⟨*fail*⟩]] [⟨*options*⟩]

The result will be a boolean value that is 1 if the d100 roll was less than or equal to ⟨*chance*⟩, and 0 otherwise. By default, the result is reported in the detailed description as "`success`" or "`fail`". If a ⟨*success*⟩ label is given in the die-roll string, that is reported in case of a successful roll, and "`did not` ⟨*success*⟩" otherwise. If an explicit ⟨*fail*⟩ label is also given, that is used for unsuccessful rolls instead. As a special case, if ⟨*success*⟩ is "`miss`" then ⟨*fail*⟩ is assumed to be "`hit`" and vice versa.

For percentile rolls, only the `until`, `repeat`, and `maximized` global options may be used. Permutations ("`{...}`") are also disallowed with percentile rolls.

Permutations

If curly braces ("`{`" and "`}`") appear anywhere in the die roll expression, with a number of slash-separated alternative values between them, the entire die roll expression will be executed once for each alternative. For example, the expression "`attack = d20 + {17/12/7} + 3`" will result in three separate rolls being performed: "`d20+17+3`", "`d20+12+3`", and "`d20+7+3`".

If multiple such sets appear in the expression, you get the Cartesian product of them all. For example, the expression "`d20+{17/12}+{d4/d8}`" results in the four rolls: "`d20+17+d4`", "`d20+17+d8`", "`d20+12+d4`", and finally "`d20+12+d8`".

Blind Rolls

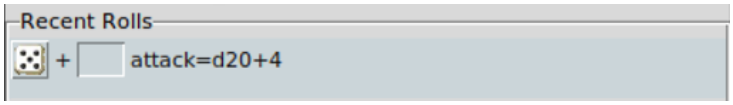
Sometimes you need to make a skill check such as Knowledge or Perception (say, to detect the presence of a secret door). The GM could roll these for you, since you shouldn't see the result of the roll yourself. (For example, you shouldn't know out of character that the natural 20 you rolled meant there really wasn't a secret door, whereas the natural 1 makes you wonder if you just overlooked it.)

But if you want to roll the dice yourself, so you can be sure all the right modifiers are in place, or just for the satisfaction of knowing you were controlling your own fate, just check the "GM" box before hitting ↵ to send your die roll. This will send it privately to the GM (regardless of the recipient setting in the chat window), and the GM is the only one who will see the results of the roll.

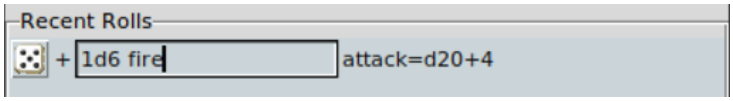
This will also affect die rolls made from the recent roll list or the presets (described below).

Re-Rolling Recent Rolls

The 10 most recent die rolls are collected in the pane under the main chat area. You can roll any of them again by clicking on the die button to the left of them.



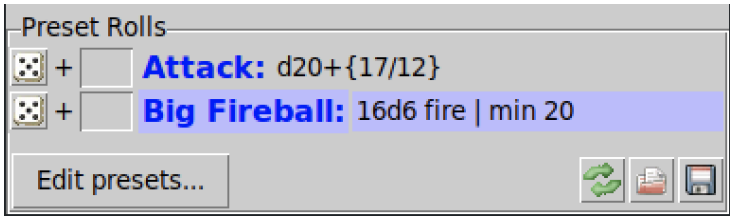
Re-rolling with a temporary modifier in place as as simple as typing the temporary addition in the box next to the die button. This may contain anything legal for a die roll expression and is appended to the original one with a "+" between them. (If your temporary modifier begins with a math operator such as "-" then no "+" is added, so you can put a penalty in that box, or other arbitrary expressions.)



Here, with the "`1d6 fire`" entered, clicking on the die button will roll "`attack = d20 + 4 + 1d6 fire`".

Pre-set Rolls

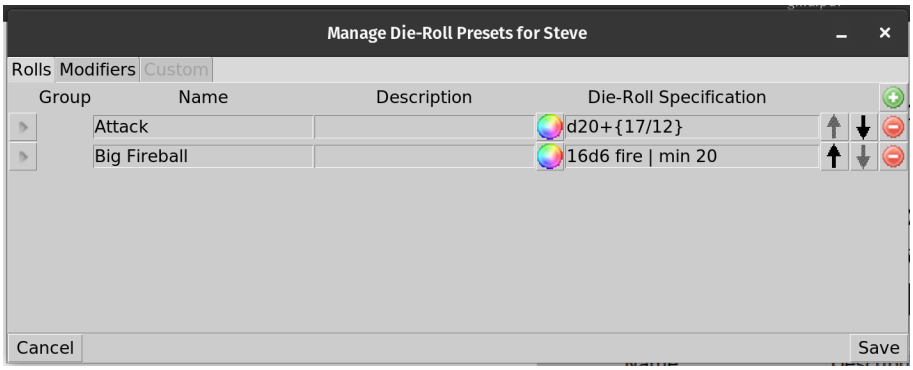
The bottom pane of the chat window lists preset rolls you have stored on the server. Every time you log in to the game server with the same user name, these will be retrieved for you to use. They are invoked (including temporary modifiers) just like recent die rolls as described above.



Clicking on the refresh ( ) button will ask the server to send you a fresh copy of your preferences which will update this display.

Clicking on will export your presets to a local disk file which you can use as a backup, to move them to another username, or to edit by hand (if you enjoy that sort of torment). Clicking on the button will import such a file back into your preset list. In this case you will be asked whether you want the imported set to be added to, or to replace, your current set of presets.

To modify the list of presets, click on the Edit presets... button at the bottom of the pane. This gives you an editor window to manage your presets:



There are three tabs in the editor. The first one is "Rolls" which manages your die-roll presets. Click on the ⊕ button to add a new preset, or the ⊖ button on the right of any existing preset to delete that preset from the list. You can click the ↑ and ↓ buttons to move a preset up or down in the list. This will affect the order in which they are displayed to you in the preset pane.[2]

The "Description" field is an optional note to yourself about what the preset is for. This will appear as a tooltip pop-up note if you hover the mouse over it in the preset pane.

If you want the die roll to have a special title string (possibly including custom colors), you can click on the ◉ button. We'll discuss how this works in a section below (see p. 203).

When you're finished with your changes to the preset list, click the Save button to commit your changes,[3] or click Cancel to abandon all the changes you made, leaving the preset list unaltered.

---

[2]Although not directly supported by the mapper's die-roll preset editor, it is possible to impose a custom ordering of the presets by inserting arbitrary text and a vertical bar (|) before the preset name (e.g., "`att000|Attack`"). The mapper will sort the preset list based on the entire name but will only display the part after the vertical bar in the pane. When using the preset editor as shown here, the mapper is managing the part to the left of the "|" for you. The "Name" entry field is the displayed name (to the right of the "|"). Behind the scenes, the mapper is prepending a three-digit sequence number ahead of your name. In the example shown here, the two presets are internally called "`000|Attack`" and "`001|Big Fireball`".

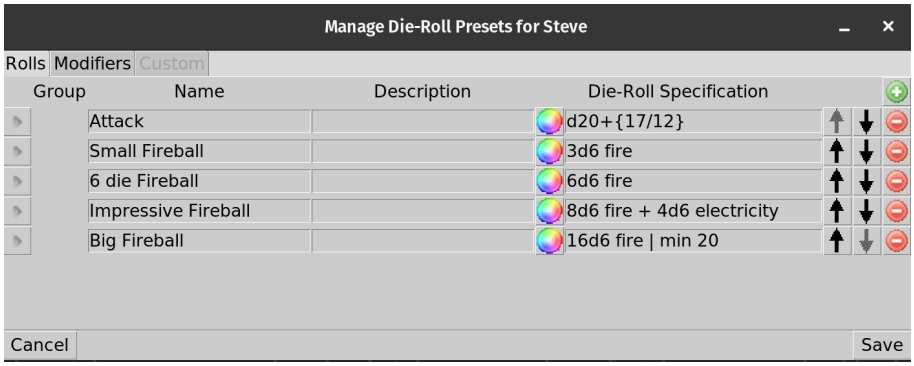
If you already have custom sorting in your die-roll presets (such as by manually editing your presets outside of the preferences editor in the mapper), and the part to the left of the bar is strictly just an integer value (or one of the expanded formats recognized by the mapper), the editor will include that in the sorted list and manage its number along with the others, renumbering it as needed. However, if you have text that is not in any recognized format, such as "`att000|...`" then the preset editor will show that preset in the "Custom" tab without interpreting or managing the name at all.

It's best not to manually edit the preset names so you don't cause confusion.

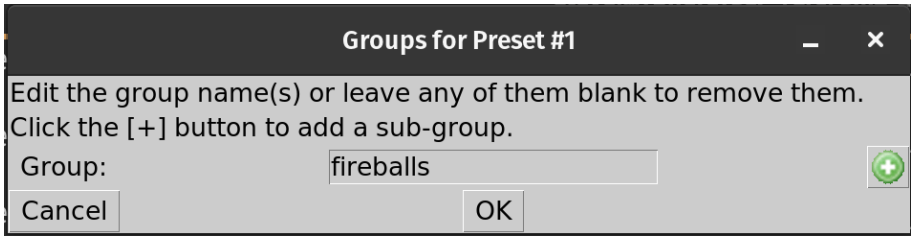
[3]They are sent directly to the server, which will store them and then send a command to your mapper (and all other mappers logged in with your username at the time) to accept the new preset list. If there's a delay between clicking Save and seeing the changes appear, it's because the data has to make that round-trip first.

Grouping Presets

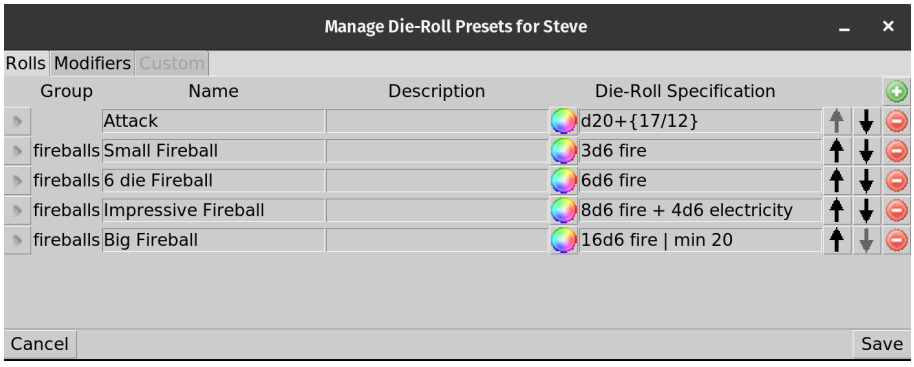
If you define a large number of presets, the scrolling list in the die roll window may get inconveniently large. To help manage this, you can assign a number of presets together into a named group. Let's say we defined a number of different fireballs.



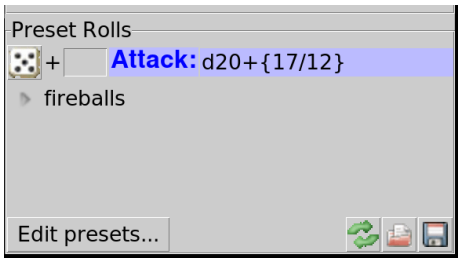
To assign one to a group, click the ▸ button to the left of that preset. This brings up a dialog to enter the group name(s) for the preset.



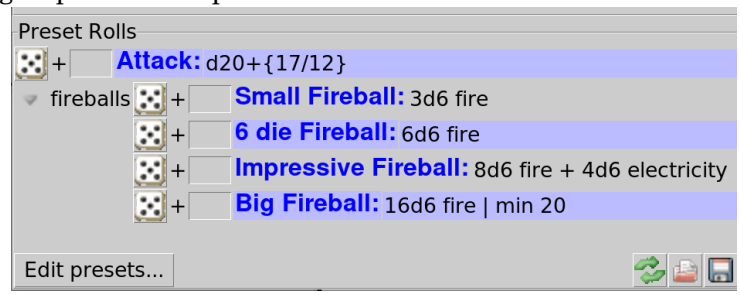
We enter the group name "`fireballs`" and click OK, adding the selected preset to that group. We repeat this for all the fireball presets.



Now when we save these changes and return to the die roller window, we see that everything in a group is collapsed away from view, with just the group name visible. All *consecutive* presets with the same group name are collapsed together as a unit.



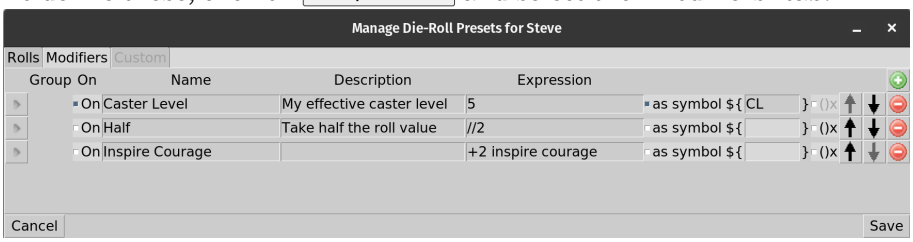
When we're ready to cast a fireball, we just click on the ▸ button next to the group name to expand them for use.



Clicking on the ▾ button will collapse them again.

Die-Roll Modifiers

Often there are temporary modifiers which may come into and out of play, or which you want to define in one location for multiple roll presets to use. To define these, click on Edit presets... and select the "Modifiers" tab.



If the "On" checkbox is selected, the modifier will be considered in play by default. When first loaded, it will be enabled in the die roller, but you can turn it on or off from there.

The first modifier has the "as symbol" checkbox selected and the name "`CL`" entered to the right of it. This means that anywhere the symbol "`${CL}`" or "`$CL`" appears in any die roll you make, it will be replaced by its value ("`5`") if the "Caster Level" modifier is "on". If the modifier is "off", then any "`${CL}`" and "`$CL`" will be removed from die rolls entirely.

This means you could have a die-roll that involves your caster level, such as "`${CL}d6 + 2*$CL`".[4]

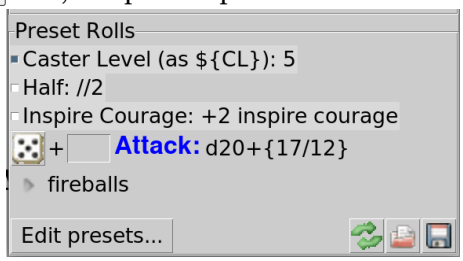
[4]Note that in this example, you couldn't use the die roll "`${CL}d6 + 2*$CL`" without the

The symbol names must begin with a letter and consist of letters and numbers for subsequent characters. They are case-sensitive (i.e., "`abc2`" and "`AbC2`" are two different symbol names).

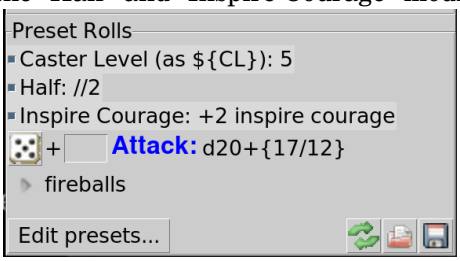
The other two modifiers shown in this example do not have symbols defined for them. This means that if the modifier is "on", it will be added to the end of *all* die rolls.[5] These will be applied in the order in which they appear in the list, so if both "Half" and "Inspire Courage" are turned on for the die roll "`d20+5`", then the resulting die roll will actually be "`(d20+5) //2 +2 inspire courage`".

The "Half" modifier has the "`()x`" checkbox selected. This means that the die-roll expression built-up so far will have a pair of parentheses placed around it before adding the modifier. Otherwise, the modifier is simply appended to the die-roll specification. This has important implications for how the expression is calculated, given the operator precedence and order of operations performed.

If we click Save now, our presets pane will look like this:



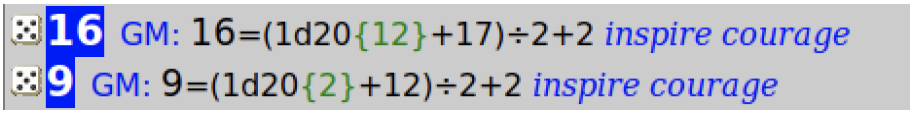
Remember, these will be applied in the order listed. If we click the "Attack" preset with the "Half" and "Inspire Courage" modifiers selected:



then the die roll will effectively become "`(d20+{17/12}) // 2 +2 inspire courage`".

Caster Level modifier on. If the modifier is off, the die roll would just be "`d6 + 2*`" which isn't valid. You could fix this by introducing a modifier (not a variable) called `x2CL` defined as "`+2*${CL}`". Then, the original die-roll would be just "`${CL}d6`". With the `CL` variable and `x2CL` modifier turned off, that would roll "`d6`". With `CL` turned on and with the value of "`5`", the roll would come out as "`5d6`"; with the `x2CL` modifier also turned on, the die roll becomes "`5d6+2*5`" or in other words "`5d6+10`".

[5] Yes, it is ok to have global modifiers in these. For example, if your die roll is "`d20+3|dc12`" and the modifier is "`+3|c18`" then the resulting die roll will be "`d20+3+3|dc12|c18`". If a global option is repeated, ones that are applied later will replace ones that appeared earlier. For example, if the die roll is "`d20|dc12`" and the modifier is "`|dc15`", that will result in a die-roll request to the server of "`d20|dc12|dc15`" which it will interpret as "`d20|dc15`".



System-Wide Presets and Modifiers

Your GM may define die-roll presets and modifiers at the system level so they are available to all players, so you don't have to individually create your own identical copies. They work the same, although you won't be able to edit or remove them.

System-wide variable names, when enabled in your preset list, may be placed in die-roll expressions by starting them with two dollar signs instead of one (this is necessary to distinguish them since it's possible that your GM defined a system-level variable with the same name as one of yours).

Thus, while you could create your own variable called CL and add it to a d20 roll by typing "d20+$CL", if your GM created a system-wide variable called bonus you could add that instead by typing "d10+$$bonus".

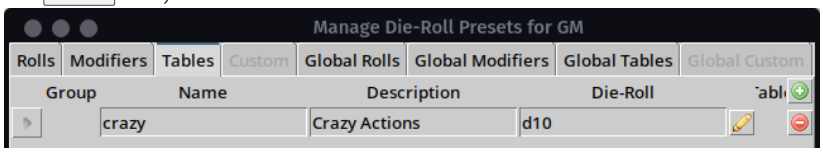
Random Lookup Tables

In many fantasy role-playing games there are cases where you want to roll dice and look up the random result on a table to see what the outcome means. For convenience, especially when making many such rolls in bulk quantities, it may be convenient to automate these table lookups. To help with this, the die roller has the ability to incorporate lookup tables by storing them in the die roll preset data along with the other die rolls and modifiers we've already discussed.

Consider, for example, a table which asks you to roll a d10 to determine the action of a creature:

d10	Action
1–5	Run around in circles
6	Do nothing
7–8	Jump up and down
9+	Run away screaming

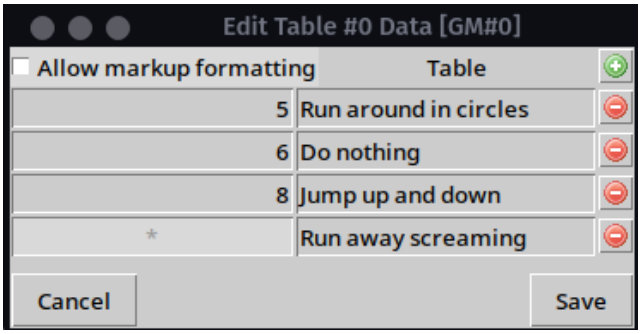
To define this table in your die-roll presets, bring up the preset editor, click on the Tables tab, and click on the button to add a new table.



Fill in the name for the table, "crazy" in this case. It should be short and consist only of alphanumeric characters. Adding a brief description of

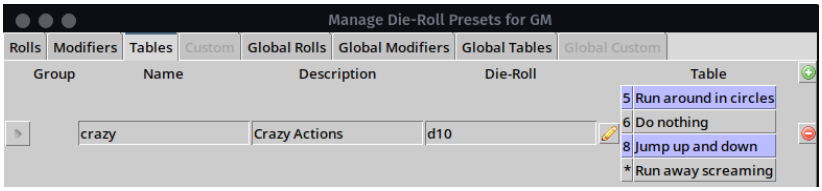
the table in the second field is recommended to help you remember what it's used for. Finally, in the third field, enter the die that needs to be rolled when consulting the table (d10 in our case here). This can be any valid die-roll expression.

Now click on the icon to edit the table data. This brings up a dialog box that allows you to add rows and fill in the columns of data into the table.

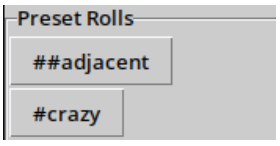


Check the "Allow markup formatting" box if you want to use GMA markup formatting codes in the text of the table itself.

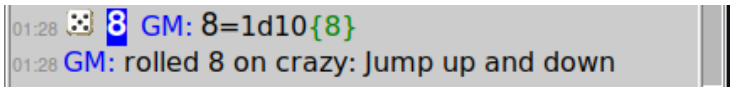
Click on the button four times to add four rows to the table. Fill in the cells of the table as needed. The left column contains the maximum die value that a given row corresponds to. So since the first row is taken for die-roll results of 1–5, we enter "5" there and put the text "`Run around in circles`" in the right column. We continue to fill in the remaining cells accordingly. We don't need a value in the left column of the last row because it'll be taken if we get that far and haven't matched any other rows yet, which covers the "9+" case. Clicking Save commits the table data and returns us to the preset editor.



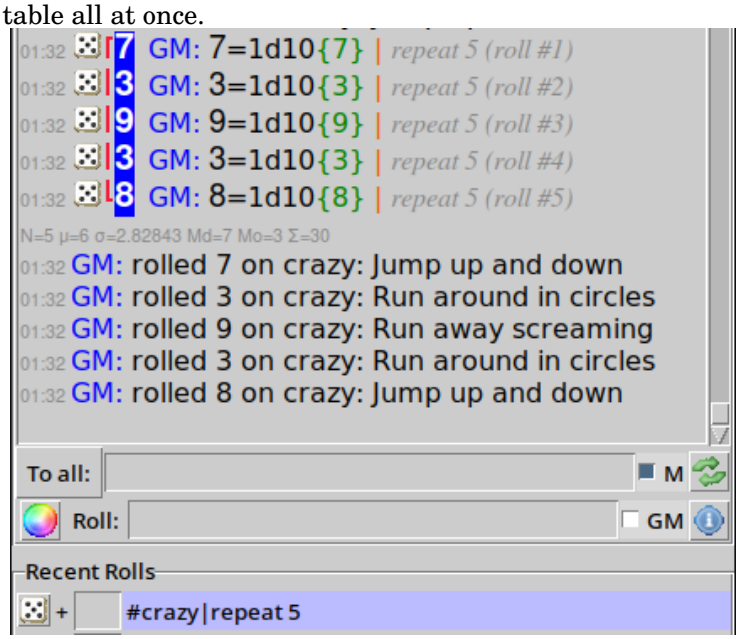
Now when we click Save here our presets will be saved back to the server and available in all our clients. Now we'll see our table in our die-roll preset list as a button we can click on.



If we do, the die roller will send a "`d10`" roll for us and then look up the result in the table, then send the meaning of that roll according to the table out as a chat message to everyone who was on the recipient list for the die roll.



Instead of clicking the button we could also have typed the table name (with a "#" in front of its name) in the die-roll entry line to get the same result. We could also follow that with other modifiers or options with the usual effects, such as "`#crazy+3`" to roll d10+3 and look that modified roll on the table or "`#crazy|repeat 5`" to roll five times on the crazy random action table all at once.



As with variables and other presets, your `GM` may define tables at the system-wide level as well, so everyone can use them. In this case, their names will have a double "`##`" prefix on their name instead of a single "`#`" one.

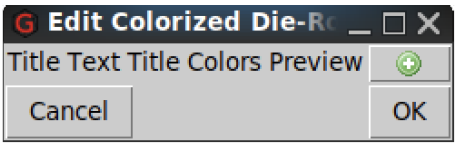
Colored Die-Roll Titles

A title may be attached to any die-roll specification by starting the specification with arbitrary text followed by an equals sign (e.g., "`Attack=d20+6`"). This will be displayed in a colored box between the die roll result and the details printed about how that result was obtained.

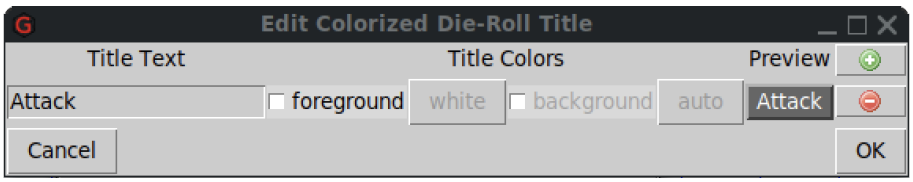
It is possible to control the color of this box, and even to have the title divided into multiple boxes, although the syntax used to accomplish this requires some Unicode characters which may be inconvenient to type manually. However, the mapper provides a tool to make this easier.



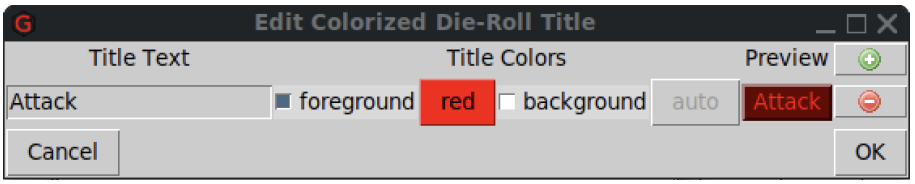
To the left of the die-roll specification input field on the mapper's chat window, and in the preset editor, you will see a button with a colored wheel icon. Click on that button to edit that die roll's title(s).



If the editor tool looks like this, that means there are no title boxes defined yet at all for this specification. Just press the button to add a title box and enter the title text.

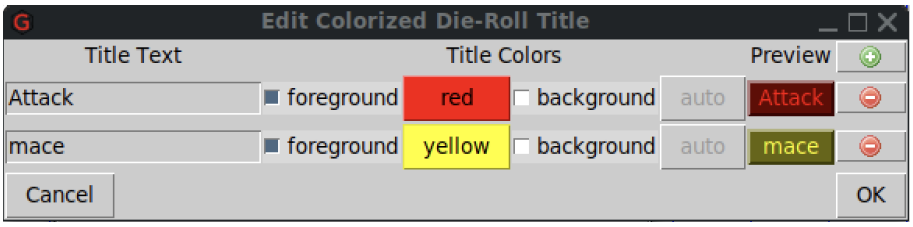


If you select the "foreground" checkbox and click on the color button to the right of it, you can select any color in which to display the title text. Notice how a background color is automatically chosen for you.



You can also select the "background" checkbox and click on its color button to change the background color to anything you want.

Clicking the button will add another title, which can be colorized in the same manner.



When you click the OK button, the die-roll specification is updated with the color information encoded into it.



Now when the die roll is executed, it will be displayed like this:

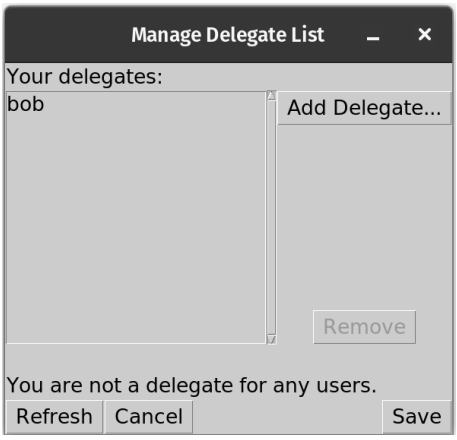


Delegating Die Rolls to Other Players

Sometimes it happens that a player can't make it to a game session, and wishes to have another player run their character in their absence. However, they may have a set of die-roll presets all set up for their character, which the substitute player can't access. What to do?

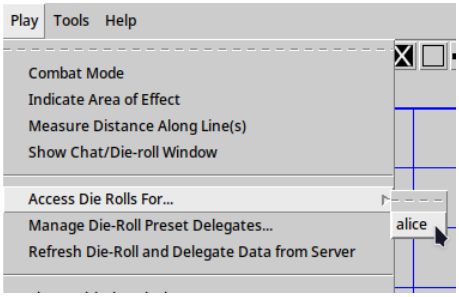
Starting with version 4.22, the mapper allows users to designate a *delegate* to manage die rolls on their behalf. Let's say Alice wants Bob to help her with rolling dice for a while. She logs in to the server with her map client, and selects the Play » Manage Die-Roll Preset Delegates... menu choice. This brings up the "Manage Delegate List" dialog. She clicks on the Add Delegate... button, enters "bob", and clicks OK. This adds Bob to her list of delegates.

Note that user names are case-sensitive. "Bob" and "bob" are two different users.



She clicks on Save. At this point, Bob has access to her die-roll settings. She can revoke this access at any time by coming back to this dialog, clicking on Bob's name in the box, then clicking Remove bob.

Now Bob can access the Play » Access Die Rolls For » alice menu choice when he needs to manage or use Alice's dice.



When he does this, he gets a die roller window specifically for Alice's dice.



This is the same as his own die roller window, except it does not have a text panel where chat messages and die-roll results are displayed (since he doesn't need multiple such windows). Any chat message or die roll made from this window, including *ad-hoc* typed expressions, recent rolls, and presets, will be noted as coming from Bob (since they are), but prefaced with the text "`(for alice)`" so it's clear he's rolling on her behalf.

If he clicks the Edit presets... button, he will be editing Alice's die-roll presets. Any changes he makes here will not only be available to Alice the next time she logs in, but if she is currently logged in, they will be updated in her die roller window in real time as well. Likewise if she edits her presets while Bob has her die roller window open.

Web-Based Die Roller

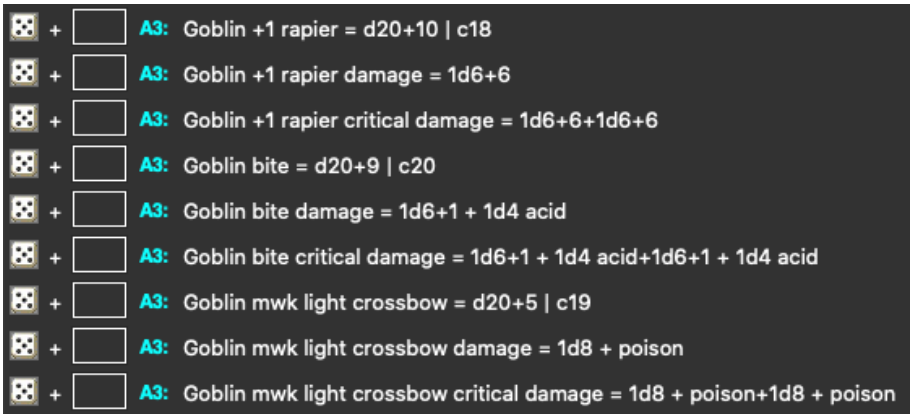
To support the case where a player cannot, or does not wish to, install any software but needs to roll dice via the GMA server, you can install the GMA web-based die roller on your web server as explained on page 788.

This allows anyone with a modern JavaScript-enabled browser to access the web interface, log in to the server, and then see chat messages and die rolls. They can make their own die rolls, typed in *ad-hoc*, or from their set of server-side presets.

Automatic Rolls for Monsters

In previous chapters we introduced increasingly automated encounter support. Now that we have the online die roller and chat capability, we can use this to our advantage in that area as well. In the initiative tracker window of GMA, when you select an encounter area and click the button, in addition to the other automatic combat setup discussed previously, this also pushes a set of attack and damage rolls to the GM's die roll preset list.

This includes attack rolls for each listed attack form or weapon from the `.gmo` file, as well as each damage roll and critical damage roll. For example, if we select the goblin encounter we used in previous examples, the GM's die-roll window gains these new presets:



The preset name for all of these generated rolls will be the name of the encounter area (e.g., "A3").

Starting with version 6.14, the `gma` program will organize these ephemeral presets into groups for easier management.

Grouping Identical Rolls

You'll note that the word "Goblin" appears in the roll presets that GMA created automatically. These are collected based on creature name in each area. What if an encounter area includes many creatures with *different* names, but identical attack forms? GMA will generate a lot of redundant die roll presets.

If the encounter includes three goblins whose image tags indicate that they are called (on the map tokens) Hinkle, Binkle, and Goldtooth, then you'll get a full set of attack rolls for each. If you'd rather generate a single set of die rolls just labelled "Goblin", you'll simply need to make a small correction to the `image_tag` for each of them to indicate that, in addition to everything else, they are all of the "Goblin" group for the purpose of making die rolls that are common to everyone in that group. This is done by putting the prefix "`Goblin::`" at the start of the `image_tag` value:

```
image_tag=Goblin::Goblin=Hinkle Goblin::Goblin=Binkle
    Goblin::Goblin=Goldtooth
```

End of Combat Cleanup

When combat is over, clicking the delete button () at the top of the initiative tracker will delete *all* of the GMA-generated die-roll presets from the GM's mapper client.

CHAPTER 11

Epic Level: Full Multi-User Server Environment

> The worthy GM never purposely kills players' PCs, He presents opportunities for the rash and unthinking players to do that all on their own.
>
> —Gary Gygax

The final level of multi-user setup is to serve all the image content from a web server. This greatly improves the speed, efficiency, and reliability of the mapper clients when you are using graphic image tiles in your maps.

Without this setup, all of the map data you load will be pushed out over the map server interface to all of the clients, which—depending on what's included on the map—can be a considerable amount of data to send over that channel.

If you're going to be using maps with images at all, this is the preferred way to set everything up.

The essential idea is that your map content consists of a (possibly quite large) collection of image files, along with a number of map files, and you need a way to efficiently serve all of them to a number of networked clients. To be efficient, it would be ideal if the clients could cache a copy of the content ahead of time, so it's ready to go at game-time. Even better if the server had a way for them to periodically check to see if a new version of

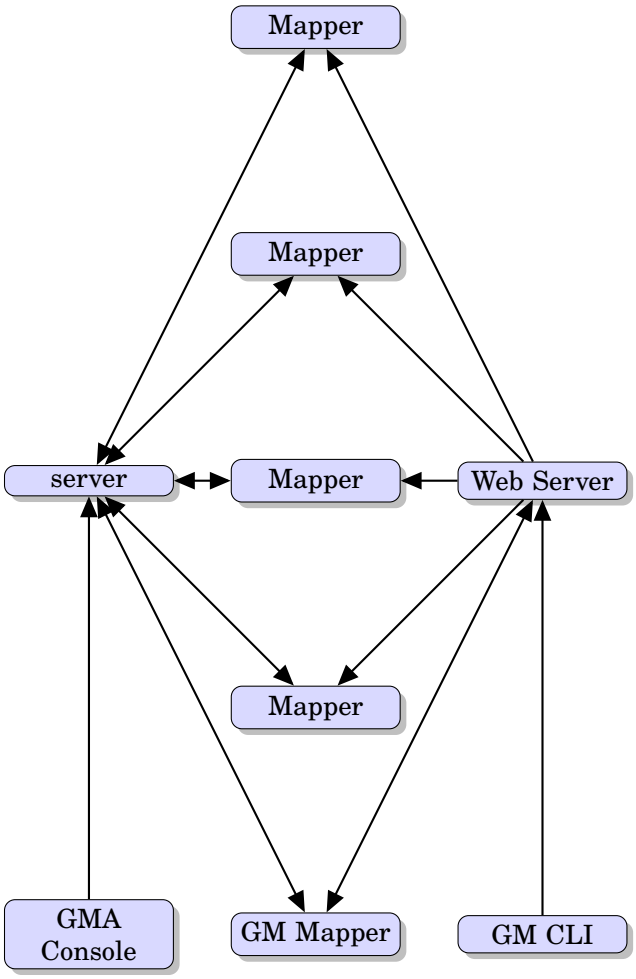


Figure 11.1: Epic-Level Multi-Player GMA Server Configuration

their files were available, and re-download them only if they've changed since they last downloaded a copy.

Fortunately, there is such a server that's readily available—a standard-issue web server (e.g., Apache).

Setting Up the Web Server

Obtain your web server of choice and follow its instructions for setting it up (this is outside the scope of this user manual, and will vary greatly depending on which server and hosting environment you use).

It will need to be hosted somewhere your players can reach from their systems. A simple, low-cost cloud hosting platform may be a great way to get a working (even pre-setup) basic web server to use for your game.

For our purposes we just need a single directory tree that belongs to GMA, that we can arbitrarily add files and subdirectories to.

Security and Permissions

You should set up all the standard security protections that are industry best practices and due diligence for internet web servers. We recommend setting the permissions of the GMA document tree so that they can't be written to except by the GM or whomever is responsible for managing your online map content.

If you choose to use some kind of password protection or authentication to control access to the map content, you'll need to have your players configure their clients to be able to provide that information. This may be overkill, however, assuming the contents of your game maps aren't terribly confidential or sensitive in nature.

Document Directory Layout

The GMA content served by your web server all exists in a single directory tree called `gma/map`. You will need to determine where on your web server to place this directory and note two things:

Document root —The location on the server of the `map` directory. For example, `/usr/local/www/data/gma/map`.

Base URL —The externally-visible URL leading to the `map` directory from a browser's point of view.

For example, `https://www.example.com/gma/map`.

All of the content GMA currently uses is in a `map` subdirectory of the top-level `gma` directory, but future features may use other subdirectories below `gma` for other things.

Go ahead and create ⟨*document root*⟩`/map` now and set its permissions so your web server can read it, and so your GM's login account to the web server can write to it, including the ability to create new directories.

This account will also need to be able to execute the `mkdir` command (or whatever its equivalent is on the operating system hosting the server) on the server system, as well as writing files into the `gma` directory tree via `ssh`/`scp`.

GM Access (Pushing Content to the Server)

The GM needs to be able to run the `mkdir` command on the server (via `ssh`) and to push files to it via `scp`, from the GMA tools. This usually involves

creating an SSH key, using that for the necessary authentication with the web server host.

If you don't have an SSH key to use, create one now. It is beyond the scope of this document to advise you on how to set up your SSH credentials, so you are directed to refer to your local documentation on the matter. For a typical system, (as an illustrative example), it will look something like this:

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/steve/.ssh/id_rsa
):
Created directory '/home/steve/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/steve/.ssh/id_rs
a.
Your public key has been saved in /home/steve/.ssh/id_rsa.pu
b.
The key fingerprint is:
SHA256:5w4+v19Yryov8CMeZtdZRdY3+57HhGcna02jKgRXQl0 steve@exa
mple
The key's randomart image is:
+---[RSA 2048]----+
|        ... .E  +|
|         . o   +o|
|          o     =|
|       . .     o |
|        S .   o..|
|        .+ . =ooB|
|        *oo + o@=|
|       +o*=  ooo=|
|       .o+=O=o. .|
+----[SHA256]-----+
```

From the pair of keys (secret and public) generated by that step (or that you already had on hand), install your public key on the web server as an authorized key to access the GMA materials in the manner already discussed.

Test it by trying to copy a local file (say, "`test.txt`") to the server using `ssh`. For example (but use the correct host and path names for your system):

```
$ scp test.txt example.com:/usr/local/www/data/gma
test.txt                      100%  683   741.1KB/s   00:00
$ ssh example.com rm /usr/local/www/data/gma/test.txt
```

If everything is correctly setup, those commands will succeed without prompting you to type your password.

Preparing Content for the Server

Recall the discussion about `rendersizes` in Chapter 5. Using the `-w` option directs `rendersizes` to generate output suitable for uploading to the server we're setting up now.

Specifically, it will generate the files into a directory hierarchy starting with a directory called `gma` in the current directory (where you're running `rendersizes`).

For example, in that chapter we generated the character token image for Agnes. If we used the `-w` option, as

```
$ gma rendersizes -Lw -f tiles.map M [#%]Agnes.png
Converting images to M-size image tiles into tiles.map...
#Agnes.png: 12 25 50 100 150 200 300 400 600 800 1600
%Agnes.png: 12 25 50 100 150 200 300 400 600 800 1600
```

this will generate image files at eleven sizes, each in three formats (PNG, GIF, and JPEG). The `tiles.map` file will also indicate that the images are not local (i.e., they are to be downloaded from a web server), and assigns them server-side filenames derived from their image names and the module ID if one was specified to `rendersizes` via the `-M` option. It creates a directory called `gma` which will hold these generated files in the expected subdirectory layout as they should be stored on the server.

For typical usage, we would probably want to have `rendersizes` generate the killed and semi-transparent images for us, and specify a module ID as well, to separate images from different modules which have the same base name. Thus, the full typical command to generate Agnes's token images would be something like:

```
$ gma rendersizes -Lwkt -M EX1 -f tiles.map M ./#Agnes.png
```

If we look at `tiles.map`, we see that the server ID for the image called `#Agnes` at zoom factor 1.0 is `IQl9xEy4zOMTvZxystwbbA_Agnes@50` (the actual name may vary depending on choice of module ID).

When the `tiles.map` file is loaded into a mapper client, it will know that it needs to ask for that server ID when it wants to display that image

at that zoom level. To do so, it will go to the web server and ask for file ⟨*base URL*⟩`/I/IQ/IQl9xEy4zOMTvZxystwbbA_Agnes@50.png`.

Once we have generated all of our images that will be used with our local maps (and we recommend following the advice in Chapter 5 about naming image files and using the `-M` option to avoid conflicts between modules on the same server), we will also use `rendersizes` to store our map files on the server where the clients can find them.

For example:

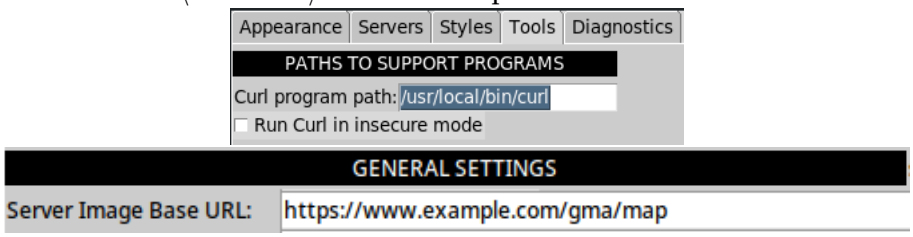
```
$ gma rendersizes -w -f maplist.map map tiles.map A*.map
Converting map files to maplist.map...
tiles.map: tiles.map -> gma/map/n/nb/nbX4yasO2xcwVR5N3PLIfw
.map
⋮
```

Now we can upload everything from our local `gma` directory tree to the web server. For example, if our documents are stored on the server at `/usr/local/www/data/gma/map`:

```
$ scp -r gma example.com:/usr/local/www/data
```

How This Works with the Mapper

In the players' mapper preferences settings, they will need to specify the location of the ⟨*base URL*⟩ as well as the path to their local `curl` executable:



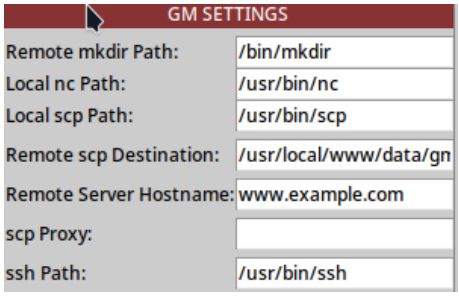
This provides the mapper with two important bits of information. First, the location (on the player's own computer) where it can find the `curl` program. It will run this program to fetch the file from the server, so if there are special command-line options needed for `curl` to naviate through proxy servers, or to provide authentication credentials, specify them in this configuration file line as well.

Second, this defines the base `URL` up through the `gma/map` directory. The client will add the rest to this for the particular file it's looking for.

Since it is often difficult to manage to have all your players' mapper configurations stay up to date, and also because there may be times you need to move the server's map data storage location (perhaps to a backup server), you can also make a server-side configuration for this base `URL`

which the mapper clients will use regardless of their local settings. See p. 221 for an example of how to configure this on your server.

For the GM, there will also need to be a few more configuration entries in the Servers tab of the preferences editor:



These are needed for pushing map content on the fly from the GM's mapper client to the server:

Remote mkdir Path —The pathname to the `mkdir` command *on the web server*. This will be executed via `ssh` when uploading content to ensure that all the required subdirectories are on the server.

Local nc Path —The pathname to the `nc` command *on the GM's local computer*.

Local scp Path —The pathname to the `scp` command *on the GM's local machine*. The mapper client will run this to transfer files to the web server.

Remote scp Destination —The pathname of the ⟨*document root*⟩ up through the `map` subdirectory where the GM's mapper client will write new data files.

Remote Server Hostname —The hostname (only) of the web server. This will be used to tell `scp` where to send data files. If you need to log in as a different user name than the one you're using on your local computer, you can specify that here as well. For example, to upload files as the user "`gm`" on host "`www.example.com`" specify this value as "`gm@www.example.com`".

If you used the `-M` option to specify a module ID for your game (e.g., "`-M GX1`") then you should also that ID on the Servers tab as well.

Example Play Using the Mapper in Store-and-Forward Mode

Having set up the server and pushed our pre-generated images and map files to it using `scp`, we're ready to play our game.

When we start up the GM mapper client, we find a new button we didn't have before on the toolbar:



The button toggles store-and-forward (SAF) mode, which is what we need to make full use of the server arrangement.

With it off (the default state), then any file we load into our mapper client is sent to clients line by line, describing each on-screen object attribute-by-attribute. If we've prepared our files ahead of time, there's only one type of file we want to load with SAF mode off: `tiles.map` and similar files that only contain the server IDs of images and other map files. Opening these files with the button broadcasts these server ID definitions out to all listening clients.

Importantly, since the server is configured to keep persistent state, it also remembers these mappings. In the future, a new client that connects to the server will ask the server if it knows where it can find any images it doesn't already know about. With this information loaded, the server will be equipped to answer those queries, allowing the client to then download the files and use them.

For the remainder of the game, as we load map files containing dungeon content, we'll want SAF mode enabled. Click on the SAF button ( ) to enable it. This will change the operation of several neighboring buttons. You can tell this is the case because they take on the marker:



Now, when you load or merge a dungeon map file using the or buttons, it loads up the map content locally but then sends a single message to the connected peers telling them to get the content from the server, supplying the server ID to use to get the map file.

Clicking on works similarly. Rather than sending all the other clients a list of objects to remove (taken from the selected file), it sends them the server ID of the file, from which they'll find the list of objects to remove from themselves.

What if the file isn't actually on the server? No problem. Before doing what we just described, the GM's mapper client first verifies that the file does, in fact, exist on the server. (If it has a recent enough version it cached from the server, it takes that as sufficient proof. Otherwise it will try to download it from the server.) If not, the mapper will first upload its local copy of the file to the server, verify that it can download it, *then* will tell the other clients to fetch it for themselves.

The button will send whatever's on the GM's map out to all the other clients. It does this by saving it to a local file, uploading it to the server, and then telling all the other clients to load it.

Client-Side Caching

Clients cache the files they obtain from the server in a local directory called `~/.gma/mapper/cache`. When a client wants to fetch a file of any kind from the server, it checks to see if it already has one in its cache.

If it does, and that file is less than 2 days old, it just uses the cached file immediately. This is designed to allow clients to drop and reconnect during a game without suddenly taking a hit to go looking around the server for updates in the middle of the action.

If the cached file is older than that, the mapper client will check on the server to see if the server's copy is newer. If not, it updates the timestamp of the cached file and proceeds with that. Otherwise, or if there is no cached file, it downloads a fresh copy from the server into the cache.

At any time, it is safe to delete the contents of this directory. The client will just repopulate it again as it needs files from the server, although this will incur the need to actually carry out the operation of downloading all of those files again.

Automatic Upgrades

The GMA project's three main products (GMA Core, GMA Mapper, and GoGMA) are available from their respective Git repositories. Interested users, including your players, may obtain their own copies by cloning those repositories.

In the case of GMA Mapper, which in most cases is the only GMA product of interest to non-GM players, they can also simply download a standalone copy of Mapper from a pre-built ZIP or tar file. These are officially released on the GMA Mapper github, where you can find the most recent several releases in the signed-releases directory. For each version available there, say version 4.3.3, there are four files:

- `mapper-4.3.3.tar.gz` (the Mapper as a compressed tar file)
- `mapper-4.3.3.tar.gz.sig` (cryptographic signature of the tar file)
- `mapper-4.3.3.zip` (the Mapper as a ZIP file)
- `mapper-4.3.3.zip.sig` (cryptographic signature of the ZIP file)

Follow the instructions in Appendix G or H to install the Mapper from either the tar or ZIP archive file.

As GM, you will probably want your group to be on the same version of the Mapper, and more importantly, one that is compatible with the version of the server you're running. You can do this by having your server advertise the desired version you want your players to be running. When their mapper clients connect, they will detect if they are not at the desired version.

Using Official Github Repository

The simplest method is to configure your clients to pull new versions of the mapper from the official Github repository for the mapper. This is sufficient if you will be using one of the most recent releases (only the most recent ones are maintained on that repository).

On the client side, have your players set their mapper preferences settings for "Mapper Upgrade URL" to "`https://raw.githubusercontent.com/MadScienceZone/gma-mapper/main/signed-releases`".

Now configure your server so that it will tell clients what version they should be running. To do this, add the following lines to your map server's initialization file (the one you name on the server's command line using the `-init-file` option):

```
UPDATES {
  "Packages": [
    {
      "Name": "mapper",
      "Instances": [
        {"Version": "4.4.1", "Token": "mapper-4.4.1"}
      ]
    }
  ]
}
```

(assuming, of course, that `4.4.1` is the mapper version being advertised here). This will direct out-of-date clients to upgrade themselves from `https://raw.githubusercontent.com/MadScienceZone/gma-mapper/main/signed-releases/mapper-4.4.1.tar.gz`.

If you are making the change to a running server, send it the `USR1` signal to have it re-read its initialization file. Otherwise, the changes will take effect the next time the server is started.

Now when a mapper client which is not as current as this advertised version connects to the server, it will prompt its user to upgrade. If the user agrees, the client will download the files from the server, verify the cryptographic signature and integrity of the files, install them, and then switch to running the new version.

Staging Your Own Clients

Alternatively, you can set up an area of your server to hold versions of the mapper software you want your players to use. In this case, have them configure their mappers' preferences to use your server URL. You can name them anything you like; the path that the clients will use to retrieve the new code will be "⟨*update-URL*⟩/⟨*token*⟩`.tar.gz`" and "⟨*update-*

URL⟩/⟨*token*⟩`.tar.gz.sig`". Set your server initialization file's `Token` value accordingly.

Create a designated directory on your web server for these files you wish to provide to your players. Let's say that is `https://www.example.com/game/updates` for the sake of this example.

Your players' mapper configuration will need to use `https://www.example.com/game/updates` for their update `URL` preferences settting.

Now that you have the `tar.gz` and `tar.gz.sig` files installed on your web server and the players' clients know where to find them, they just need to know that an upgrade is available. To do this, add the following lines to your map server's initialization file (the one you name to the server's command line using the `-init-file` option):

```
UPDATES {
  "Packages": [
    {
      "Name": "mapper",
      "Instances": [
        {"Version": "4.3.3", "Token": "XYZZY42"}
      ]
    }
  ]
}
```

(assuming, of course, that `4.3.3` is the mapper version being advertised here, and that you named them `XYZZY42.tar.gz` and `XYZZY42.tar.gz.sig`).

If you are making the change to a running server, send it the `USR1` signal to have it re-read its initialization file. Otherwise, the changes will take effect the next time the server is started.

Only one version of each program may be listed here for each type of system, so if you already had an `UPDATES` entry for `mapper` in your server initialization file, you would replace the `Version` and `Token` values with those appropriate for the version of Mapper you want your players to use.

You can list different files for different operating systems or hardware platforms if that is necessary, like so:

```
UPDATES {
  "Packages": [
  {
    "Name": "mapper",
    "Instances": [
    {
      "Version": "4.3.3",
      "Token": "mapper_any_4.3.3"
    },
    {
      "Version": "4.3.3",
      "OS": "windows",
      "Token": "mapper_windows_4.3.3"
    },
    {
      "Version": "4.3.2",
      "OS": "freebsd",
      "Arch": "amd64",
      "Token": "mapper_bsd_64_4.3.2"
    },
    {
      "Version": "4.3.2",
      "OS": "freebsd",
      "Token": "mapper_bsd_32_4.3.2"
    }
    ]
  }
  ]
}
```

With this configuration, Mapper version 4.3.2 will be advertised for FreeBSD systems from the file `mapper_bsd_64_4.3.2.tar.gz` on 64-bit x86 architecture systems, or from the file `mapper_bsd_32_4.3.2.tar.gz` for all other FreeBSD systems; Mapper version 4.3.3 will be advertised from the file `mapper_windows_4.3.3.tar.gz` for Windows systems. Clients on any other platforms and architectures will download `mapper_any_4.3.3.tar.gz`.

The order in which the different instances of a package are listed here does not matter. The Mapper client will choose the one that most closely matches its operating system and hardware architecture from those advertised by the server.

Server-Side Client Configuration

There are a number of client preferences which can be set in your server's initialization file, which will then be sent to all clients when they connect to the server. These will override whatever the user's local preferences were (if any). This allows you to more effectively manage where mappers and other clients go to retrieve map and image content during game play.

For example, in order to retrieve an image tile to display on the map, the client needs to know the base `URL` from which it can be obtained. To set this globally for all clients connecting to this server, place a `WORLD` command with a `ClientSettings` value in your server's initialization file:

```
WORLD {
   "Calendar": "golarion",
   "ClientSettings: {
      "ImageBaseURL": "https://www.example.com/gma"
   }
}
```

The full set of server-side configuration values may be found in the server documentation starting on p. 597.



COMMAND-LINE UTILITY REFERENCE

COMMAND
School enchantment (compulsion)
Level cleric 1
Casting Time 1 standard action
Components V
...
You give the subject a single command, which it obeys to the best of its ability at its earliest opportunity.... If the subject can't carry out your command on its next turn, the spell automatically fails.

—Pathfinder *Core Rulebook*

THE FOLLOWING PAGES provide the full details for the GMA command-line (CLI) utilities. These are also available on the system where GMA is installed in the form of online manual pages ("manpages"), which can be viewed using the `man` command of GMA (e.g., "`gma man printchar`").

These entries are arranged into *sections*, as indicated by the number which follows the entry name, as in "`EntryName`(1)." The section numbers are:

1: Command-line utility programs (this appendix).

3: API library functions.

5: File formats.

6: Games.

7: Special Files and Protocols.

gma-lockfile(1)

NAME

gma lockfile – Simple lock file manager (CLI interface)

SYNOPSIS

`gma lockfile` [`-hqv`] [`-c` ⟨*comment*⟩] [`-f` ⟨*secs*⟩] [`-m` ⟨*mode*⟩] [`-t` ⟨*secs*⟩] ⟨*files*⟩ ...
`gma lockfile` [`--help`] [`--comment` ⟨*comment*⟩] [`--force` ⟨*secs*⟩] [`--mode` ⟨*mode*⟩] [`--query`] [`--timeout` ⟨*secs*⟩] [`--verbose`] ⟨*files*⟩ ...

DESCRIPTION

This provides a command-line interface to the facility implemented in the GMA API's `MadScienceZone.Common.LockFile` module (see `gma-lockfile`(3)).

For each of the ⟨*files*⟩ listed on the command line, this program attempts to create that file as a lockfile, providing another process didn't beat us to creating one.

OPTIONS

`-c`, `--comment` ⟨*comment*⟩
: Write the ⟨*comment*⟩ text into the lockfile.

`-f`, `--force` ⟨*secs*⟩
: If the timeout period expires and the existing lockfile is still there, force its deletion (and then create our own), provided the file is older than ⟨*secs*⟩ seconds.

`-h`, `--help`
: Print a summary of these command-line options and exit.

`-m`, `--mode` ⟨*mode*⟩
: Create the lock file with the specified octal ⟨*mode*⟩ value. The default is 644.

`-q`, `--query`
: If invoked with this option, it will only test for the existence of lock files, not create its own. If any of the named ⟨*files*⟩ exists, a description of the lock is printed to standard output and the return code from this program will be 20. If none are present, messages to that effect will be printed and the return code will be 0.

`-t`, `--timeout` ⟨*secs*⟩
: Wait up to ⟨*secs*⟩ seconds for an existing file to disappear before giving

up. If, in that time, the file is deleted (i.e., the lock it represents is released by its process), we will create ours and move on. The default is 60 seconds.

`-v`, `--verbose`
: Print extra verbose messages to standard output.

This program does not remove lock files (except as noted above). When the calling process no longer needs them, it should simply remove the files.

SEE ALSO

`gma-lockfile`(3).

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

APPENDIX B

API Function Reference

> Some books are not meant to be read, and some people dedicate their lives to prevent others from reading such forbidden books. Sometimes such dedication extends beyond life.
>
> —"Forbidden Library" entry, *Terrors of Obsidian Apocalypse: Haunts* by Louis Porter Jr. Design

The following pages provide the full details for the GMA API functions available to programmers wishing to write new programs which are part of the GMA tool set.

In each case, an entry is made for each individual function unless a group of related functions are so close in their description that it makes more sense to describe them as a group. In the latter case, a "see also" reference is included for all of the function names which point back to the main entry where they are documented.

Note that we follow traditional notation for function documentation where a pair of parentheses are added to the end of a function or method name to distinguish it from a variable or object attribute name. Thus, "`foo`" is a variable, member, or attribute name while "`bar()`" is a function or method name. This does not mean there are no arguments expected to be passed to the function when invoked.

These manual entries are also available on the system where GMA is installed in the form of online manual pages ("manpages"), which can be viewed using the `man` command of GMA (e.g., "`gma man 3 dice`"). These pages are in section 3, so following the conventions for Unix manpages, their names have "(3)" appended to them.

We don't include documentation for the Go language API functions here, however. Following the usual convention for Go module documentation, those may be viewed at https://pkg.go.dev/github.com/MadScienceZone/go-gma/v5.

gma-abilityscore(3)

NAME

AbilityScore – Creature ability scores

SYNOPSIS

```
import MadScienceZone.GMA.AbilityScore
⟨ability⟩ = AbilityScore(⟨base⟩, ⟨temp⟩=None)
⟨ability⟩.base
⟨ability⟩.temp
⟨mod⟩ = ⟨ability⟩.mod(⟨of⟩=None)
⟨score⟩ = ⟨ability⟩.score()
```

DESCRIPTION

A creature's basic ability scores (strength, dexterity, etc.) are each represented by an `AbilityScore` object. The constructor fills in the object's attributes and they may be changed as needed after that.

⟨*ability*⟩`.base`
: The base ability score.

⟨*ability*⟩`.temp`
: If there is some temporary effect that changes this ability score, the temporary score is stored here, otherwise this has the value `None`.

The current ability score is available via the ⟨*ability*⟩`.score()` method, which returns the `temp` value if there is one; otherwise it returns the `base` score.

The ⟨*ability*⟩`.mod()` method returns the modifier corresponding to the base (or temporary, if there is one) ability score. For example, if the ability score value ⟨*ability*⟩`.base` were 12, then ⟨*ability*⟩`.mod()` would return 1, since a score of 12 corresponds to a +1 bonus to rolls involving this ability.

This method can also be used to calculate the modifier for any ability score value rather than the one tracked by this object. So regardless of the score in ⟨*ability*⟩, calling ⟨*ability*⟩`.mod(14)` would return 2, and ⟨*ability*⟩`.mod(8)` would return –1.

SEE ALSO

`gma-Character`(3), `gma-Monster`(3).

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

gma-alignment(3)

NAME

Alignment – Creature alignment

SYNOPSIS

```
import MadScienceZone.GMA.Alignment
⟨align⟩ = Alignment(⟨align_code⟩)
⟨align⟩.code
⟨align⟩.name
⟨align⟩.bincode

⟨iter⟩ = Alignment.all_alignments()

⟨code⟩ = Alignment.code_for(⟨name⟩)
```

DESCRIPTION

The alignment of a creature is represented by an `Alignment` object. Each alignment has a one- or two-letter code, namely `CN`, `LN`, `LG`, `NG`, `CG`, `LE`, `NE`, `CE`, or `N`. It also has a human-readable name such as "Chaotic Neutral" or "Lawful Good".

A new object is constructed by passing the alignment's code to the constructor, as in: ⟨*align*⟩ `= Alignment(CG)`. The following attributes may be read from the object once created:

⟨*align*⟩`.code`
: The alignment code such as "`LG`".

⟨*align*⟩`.name`
: The alignment name such as "Lawful Good".

⟨*align*⟩`.bincode`
: A binary code value for the alignment. This supports the system of representing all aspects of a character in a binary code, which never really caught on and may be deprecated later.

In addition, there are two class methods to get information about alignments:

⟨*iter*⟩`=Alignment.all_alignments()`
: This is an iterator yielding all defined alignments. Each is returned as a tuple of (⟨*code*⟩, ⟨*name*⟩).

⟨*code*⟩=`Alignment.code_for(`⟨*name*⟩`)`
: Given an alignment name, returns the code for that alignment. The name is matched regardless of case. If no such name is found, a `KeyError` exception is raised.

SEE ALSO

`gma-Character`(3), `gma-Monster`(3).

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

gma-all_culture_names(3)

See gma-pfnamegenerator(3), p. 306.

gma-armour(3)

NAME

Armour – Keeping creatures protected from harm

SYNOPSIS

```
import MadScienceZone.GMA.Armour
Armour.core_data

Armour.quality_codes

⟨armr⟩ = Armour(⟨code⟩, ⟨size_index⟩=4, ⟨cost⟩=None, ⟨ac⟩=None,
⟨maxdex⟩=None, ⟨chk⟩=None, ⟨arcanefail⟩=None, ⟨speed⟩=None,
⟨wt⟩=None, ⟨qualities⟩=None, ⟨name⟩=None, ⟨on⟩=False)
⟨armr⟩.ac
⟨armr⟩.arcanefail
⟨armr⟩.chk
⟨armr⟩.code
⟨armr⟩.cost
⟨armr⟩.maxdex
⟨armr⟩.name
⟨armr⟩.on
⟨armr⟩.qualities
⟨armr⟩.size
⟨armr⟩.speed
⟨armr⟩.wt
⟨dict⟩ = ⟨armr⟩.overrides()
```

DESCRIPTION

A piece of armor is modelled by an `Armour` object as described here. The constructor takes all the various attributes which are significant to describe a piece of armor:

⟨*code*⟩
: The unique code name by which this item is known in the `GMA` core database. These codes are generally derived in an obvious way from the name of the armor, such as "`HideShirt`" or "`StuddedLeather`". If ⟨*code*⟩ is not found in the core database, an `InvalidArmourCode` exception is raised.

  This is the only required parameter to the constructor. All attributes of the armor are loaded from the core data. If any other parameters are given, they will override what is in the core data, to allow customizations of specific items.

If you need to build a piece of armor completely from scratch, use "`_special`" as the ⟨*code*⟩.

⟨*size_index*⟩
: The size of creature the armor is designed for. The default is medium size. This will automatically scale cost and weight unless those values are explicitly given as parameters. The size is given as a numeric value:

Index	Size
0	F (fine)
1	D (diminutive)
2	T (tiny)
3	S (small)
4	M (medium)
5	L (large)
6	H (huge)
7	G (gargantuan)
8	C (colossal)

⟨*cost*⟩
: The market price in copper pieces (so the value can be expressed as an integer).

⟨*ac*⟩
: The armor class modifier provided by the armor. This will assumed to be of type `armr` unless the ⟨*qualities*⟩ parameter (or core data) indicates that this is a shield or natural armor, in which case it will be of type ⟨*shld*⟩ or ⟨*natl*⟩ respectively.

⟨*maxdex*⟩
: gives the maximum dexterity modifier the creature may have while wearing the armor or `None` if not applicable.

⟨*chk*⟩
: is the penalty to armor checks imposed by this armor, or `None` if there is none.

⟨*arcanefail*⟩
: is the percentage chance (as an integer) that arcane spells will fail if cast while wearing this armor, or `None` if not applicable.

⟨*speed*⟩
: is a tuple of two values giving the top speed allowed for creatures whose unarmored speed is 30' and 20' respectively, or `None` if not applicable.

⟨*wt*⟩
: the weight in grams of the armor.

⟨*qualities*⟩
: Special armor qualities, as a string of characters, one character for each quality, with the following meanings:

Code	Quality
`E`	elven
`G`	gnome
`H`	halfling
`h`	heavy
`L`	light
`m`	medium
`N`	natural
`O`	orc
`s`	shield
`W`	dwarven
`Z`	masterwork
`$`	special
`-`	(ignored; may be used for formatting)

⟨*name*⟩
: The name of the armor as would be written on a character sheet.

⟨*on*⟩
: A boolean value. If true, the armor is being worn now by the creature.

These are visible as attributes of the object with the same names, with the following differences:

⟨*ac*⟩
: This is a dictionary mapping armor type code to the modifier provided for that type. For example, `{'armr': +2}`.

⟨*qualities*⟩
: This is a set of quality strings as listed in the table above. For example, `set('light','elven')`.

⟨*size*⟩
: This is the single-character size category code such as `S` or `L`.

When writing a description of an item to a data file such as the character record file, it is desirable to only write the armor code along with any variations from the core data, so that corrections in the core data will still apply to that saved data. Calling the ⟨*armr*⟩`.overrides()` method returns a dictionary of attribute names and values which differ from what is in the core data.

CORE DATA

Someday the core data should be moved to some sort of database, but for now the built-in core data for all known types of armor is stored in the class definition as the class attribute `Armour.core_data`. This is a simple dictionary mapping armor `ID` to a tuple of (⟨*cost*⟩, ⟨*ac*⟩, ⟨*maxdex*⟩, ⟨*chk*⟩, ⟨*arcanefail*⟩, ⟨*speed*⟩, ⟨*wt*⟩, ⟨*qualities*⟩, ⟨*name*⟩).

The class variable `Armour.quality_codes` lists the valid quality codes.

SEE ALSO

`gma-Character`(3), `gma-Monster`(3).

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

BUGS

The `overrides()` method does not yet save qualities.

gma-authenticator(3)

NAME

Authenticator – GMA Mapper Service Authenticator

SYNOPSIS

```
import MadScienceZone.GMA.Mapper.Authentication

⟨auth⟩ = Authenticator(⟨secret⟩=None, ⟨client⟩=None,
⟨gm_secret⟩=None, ⟨username⟩=None)
⟨challenge⟩ = ⟨auth⟩.generate_challenge()
⟨challenge⟩, ⟨iterations⟩ =
⟨auth⟩.generate_challenge_with_iterations()
⟨auth⟩.accept_challenge(⟨challenge⟩, ⟨iterations⟩=0)
⟨response⟩ = ⟨auth⟩.generate_response()
⟨auth⟩.validate_response(⟨response⟩)
⟨auth⟩.reset()
⟨username⟩, ⟨client⟩ = ⟨auth⟩.client_info()
```

DESCRIPTION

The challenge/response negotiation between the mapper server and clients is implemented in this object. A client wishing to authenticate should create an authenticator as

⟨*auth*⟩ = `Authenticator(`⟨*secret*⟩`,` ⟨*client*⟩`, username=`⟨*username*⟩`)`

where ⟨*secret*⟩ is the shared secret (password) used by clients to authenticate to the server, and ⟨*client*⟩ is a string describing what sort of client this is. If the ⟨*username*⟩ parameter is passed, it will be sent to the server to identify the user (as shown in chat messages, etc.). Otherwise, the username will be taken from their local computing environment.

Servers should create their authenticator object as

⟨*s_auth*⟩ = `Authenticator(`⟨*secret*⟩`, gm_secret=`⟨*gm_secret*⟩`)`

where ⟨*secret*⟩ is as described above, and ⟨*gm_secret*⟩ is the password used exclusively by the GM.

The calculation method used for this authentication mechanism is documented in the protocol specification in gma-mapper-protocol(7).

Note that the ⟨*secret*⟩ and ⟨*gm_secret*⟩ parameters are `bytes` values. If you have a password as a regular string value, you'll need to convert it to `bytes` (such as by calling its `encode()` method).

Negotiation Process

On the server side, an `Authenticator` object ⟨*s_auth*⟩ is constructed for each incoming connection. It issues a challenge unique to its session, and expects to see the corresponding response to that challenge, while other simultaneous incoming connections have their own ⟨*s_auth*⟩ objects negotiating with their incoming clients.

The server calls ⟨*s_auth*⟩`.generate_challenge_with_iterations()` to create a random challenge and iteration count, which are sent to the client.

The client then calls ⟨*auth*⟩`.accept_challenge()` with the challenge and iteration count it received from the server, and then calls ⟨*auth*⟩`.generate_response()` to calculate the expected response. This is sent to the server.

The server then calls ⟨*s_auth*⟩`.validate_response()` to determine if the response was valid.

Methods

The following methods are available for any `Authenticator` object:

⟨*challenge*⟩=⟨*auth*⟩`.generate_challenge()`

(server) Generate a random challenge for the client. This is stored internally for later validation. Returns a base-64 encoded representation of the challenge.

This method is deprecated. Use `generate_challenge_with_iterations` instead, or read the ⟨*auth*⟩`.iterations` attribute after the method call to get the iteration count to send to the client.

⟨*challenge*⟩, ⟨*iterations*⟩=⟨*auth*⟩`.generate_challenge_with_iterations()`

(server) Generate a random challenge and iteration count for the client. These are stored internally for later validation. Returns a base-64 encoded representation of the challenge and an integer iteration count.

⟨*auth*⟩`.accept_challenge(`⟨*challenge*⟩, ⟨*iterations*⟩`=0)`

(client) Accept a server's challenge as a base-64 string, storing it internally. You must also pass the iteration count as given by the server as the ⟨*iteration*⟩ parameter. This is required starting with server protocol 415. If you don't specify it, or give the value 0, then old protocol behavior is used (up to 414). This derives the iteration count from the first two bytes of the challenge.

⟨*response*⟩=⟨*auth*⟩`.generate_response()`

(client) Generate a response to the previously-accepted challenge. This is returned as a base-64 encoded string value.

⟨*bool*⟩=⟨*auth*⟩`.validate_response(`⟨*response*⟩`)`

(server) Returns `True` if the ⟨*response*⟩ value matches the response

expected for the challenge this object had previously generated. If the response was created using the GM's password, then the ⟨*auth*⟩`.gm_mode` attribute is set to `True`, otherwise it will be `False`.

⟨*auth*⟩`.reset()`
: Resets the internal state of the ⟨*auth*⟩ object, so it may be used to negotiate a new authentication.

⟨*user*⟩,⟨*client*⟩=⟨*auth*⟩`.client_info()`
: Returns the username and client currently in use for this authenticator. Nothing in the negotiation provides these; they must have been given to the constructor (or, in the case of the username, determined by looking at the system environment).

SEE ALSO

`gma-socketinterface`(3), `gma-mapper`(6).

AUTHOR

Steve Willoughby / steve@madscience.zone.

gma-centimeters_to_feet(3)

See gma-weightsandmeasures(3), p. 351.

gma-class(3)

NAME

Class – Character class

SYNOPSIS

```
import MadScienceZone.GMA.Class
ClassAbility.core_data
Class.core_data
Class.core_xp_data

⟨code⟩ = Class.code_for(⟨class⟩)

⟨ca⟩ = ClassAbility(⟨ability_name⟩, ⟨extra_args⟩=None,
⟨min_level⟩=None)
⟨ca⟩.ref ⟨ca⟩.code ⟨ca⟩.ctype ⟨ca⟩.feat_type ⟨ca⟩.for_class
⟨ca⟩.spellcasting ⟨ca⟩.exploit_info
⟨text⟩ = ⟨ca⟩.description_text(⟨sublist⟩, ⟨safe_mode⟩=False)
⟨text⟩ = ⟨ca⟩.full_description_text(⟨sublist⟩,
⟨safe_mode⟩=False)
⟨dict⟩ = ⟨ca⟩.modifiers_for(⟨activity_list⟩, ⟨sub_dict⟩=None,
⟨local_data⟩=None)
⟨value⟩ = ⟨ca⟩.resolve_mods(⟨value⟩, ⟨sub_dict⟩, ⟨local_data⟩)

⟨c⟩ = Class(⟨class_code⟩, ⟨level⟩, ⟨special_dict⟩=None)
⟨c⟩.code
⟨c⟩.level
⟨c⟩.update_feat_data()
⟨level⟩ = ⟨c⟩.xp_at_level(⟨level⟩=None)
⟨mod⟩ = ⟨c⟩.save_base(⟨save_type⟩, ⟨level⟩=None)
```

DESCRIPTION

The `Class` class (**cough**) describes a character class that a creature has earned. The class method `Class.all_classes()` returns an iterator that yields a list of all defined classes, each represented by a tuple `(`⟨*code*⟩`,` ⟨*name*⟩`)`. Similarly, the method `Class.code_for(`⟨*name*⟩`)` returns the class code that corresponds to the class with the given ⟨*name*⟩. This is matched regardless of case. If no such class is found, a `KeyError` exception is raised.

The constructor is called as

⟨*c*⟩ = `Class(`⟨*class_code*⟩`,` ⟨*level*⟩`,` ⟨*special_dict*⟩`=None)`

where

⟨*class_code*⟩
: is the one- or two-letter code that corresponds to the class, such as `W` for Wizard or `F` for Figthter.

⟨*level*⟩
: is the level of experience the creature has achieved in this class.

⟨*special_dict*⟩
: is an optional dictionary mapping the code for a feat or class ability to a vertical-bar-delimited list of values which are needed to fully specify this character's use of that ability.

The attributes of the new object are set from the core data loaded and include

⟨*c*⟩`.code`
: The class code for this class.

⟨*c*⟩`.level`
: The level of experience attained in this class.

⟨*c*⟩`.special_dict`
: As described above, although this shouldn't be accessed directly.

⟨*c*⟩`.CL_adj`
: Class offset from class level.

⟨*c*⟩`.CL_mod`
: User adjustment to arrive at effective class level (which will be ⟨*c*⟩`.level`+⟨*c*⟩`.CL_adj`+⟨*c*⟩`.CL_mod`)

⟨*c*⟩`.bab`
: The base attack bonus for this class based on the current level. (This value is dynamically computed every time it is accessed.)

⟨*c*⟩`.fort`
: The fortitude saving throw bonus for this class based on the current level. (This value is dynamically computed every time it is accessed.)

⟨*c*⟩`.refl`
: The reflex saving throw bonus for this class based on the current level. (This value is dynamically computed every time it is accessed.)

⟨*c*⟩`.will`
: The will saving throw bonus for this class based on the current level. (This value is dynamically computed every time it is accessed.)

⟨*c*⟩`.feats`
: A list of all class abilities and feats

The `ClassAbility` object is a subclass of `Feat`, but is used for class features. These are functionally very similar to feats, and are treated the same by GMA for the most part, but since they are acquired in a different manner and associated with progress along a particular class, they are defined in this module and have their own class.

In addition to the attributes and methods described in `gma-Feat`(3), the constructor may be given a ⟨*min_level*⟩ parameter, which will set (or override) the ⟨*min_level*⟩ attribute of the class ability in question. Otherwise the ability is loaded from the built-in core data in the same manner as for feats.

SEE ALSO

`gma-Character`(3), `gma-Feat`(3), `gma-Monster`(3).

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

gma-culture_class(3)

See gma-pfnamegenerator(3), p. 306.

gma-default_core_db_path(3)

See gma-preferences(3), p. 310.

gma-default_preferences_path(3)

See gma-preferences(3), p. 310.

gma-dice(3)

NAME

Dice – Dice Rolling Functions

SYNOPSIS

```
import MadScienceZone.GMA.Dice

⟨d⟩ = Dice(⟨qty⟩=1, ⟨sides⟩=20, ⟨bonus⟩=0, ⟨diebonus⟩=0,
⟨div⟩=None, ⟨description⟩=None, ⟨factor⟩=None, ⟨generator⟩=None)
⟨result⟩ = ⟨d⟩.roll(⟨confirm⟩=False, ⟨threat⟩=None, ⟨bonus⟩=None)
⟨result⟩ = ⟨d⟩.value()
⟨result⟩ = ⟨d⟩.maxvalue()
⟨str⟩ = ⟨d⟩.description()
⟨str⟩ = ⟨d⟩.describe_roll()
⟨structured_results⟩ = ⟨d⟩.structured_describe_roll(⟨sf⟩=None)

⟨dr⟩ = DieRoller(⟨generator⟩=None)
⟨label⟩, ⟨result_set⟩ = ⟨dr⟩.do_roll(⟨dice⟩=)
⟨str⟩ = ⟨dr⟩.to_text(⟨structured_results⟩, ⟨label⟩=None,
⟨formatting⟩=None)

⟨pc⟩ = DieRollPresetCollection([⟨iterable⟩,] ⟨timestamp⟩=0,
⟨date_time⟩=None, ⟨comment⟩=None)
⟨pc⟩.set_timestamp(⟨timestamp⟩=0)
⟨pc⟩.write_preset_file(⟨path⟩)
⟨pc⟩.save_preset_file(⟨f⟩)
⟨pc⟩.read_preset_file(⟨path⟩)
⟨pc⟩.load_preset_file(⟨f⟩)
⟨pc⟩.append(⟨preset⟩)
⟨oc⟩=⟨pc⟩.get_details()

⟨oc⟩ = DieRollObjectCollection([⟨iterable⟩,] ⟨timestamp⟩=0,
⟨date_time⟩=None, ⟨comment⟩=None)
⟨oc⟩.write_preset_file(⟨path⟩)
⟨oc⟩.save_preset_file(⟨f⟩)
⟨oc⟩.read_preset_file(⟨path⟩)
⟨oc⟩.load_preset_file(⟨f⟩)
⟨pc⟩=⟨oc⟩.encode()
⟨oc⟩.resequence()

⟨pre⟩ = DieRollPreset(⟨name⟩, ⟨description⟩, ⟨die_roll_spec⟩)
⟨pre⟩.from_tuple((⟨name⟩, ⟨description⟩, ⟨die_roll_spec⟩))
⟨tuple⟩ = ⟨pre⟩.to_tuple()
⟨dictionary⟩ = ⟨pre⟩.to_dict()

⟨obj⟩ = get_preset_details(⟨pre⟩)

⟨s_obj⟩ = SimplePreset(⟨name⟩, ⟨description⟩, ⟨die_roll_spec⟩,
⟨seq⟩=None, ⟨group⟩=None, ⟨flags⟩=None, ⟨client_data⟩=None)
⟨pre⟩ = ⟨s_obj⟩.encode()
```

```
⟨d_obj⟩ = DictPreset(⟨encoded_name⟩, ⟨description⟩,
⟨die_roll_spec⟩, ⟨seq⟩=None, ⟨group⟩=None, ⟨client_data⟩=None)
⟨d_obj⟩ = DictPreset.New(⟨name⟩, ⟨description⟩='',
⟨die_roll_spec⟩='{}', ⟨seq⟩=None, ⟨group⟩=None,
⟨client_data⟩=None)
⟨pre⟩ = ⟨d_obj⟩.encode()

⟨m_obj⟩ = DieRollModifier(⟨encoded_name⟩, ⟨description⟩,
⟨die_roll_spec⟩, ⟨seq⟩=None, ⟨group⟩=None, ⟨varname⟩=None,
⟨flags⟩=None, ⟨client_data⟩=None)
⟨m_obj⟩ = DieRollModifier.New(⟨name⟩, ⟨description⟩,
⟨die_roll_spec⟩, ⟨seq⟩=None, ⟨group⟩=None, ⟨varname⟩=None,
⟨flags⟩=None, ⟨client_data⟩=None)
⟨pre⟩ = ⟨m_obj⟩.encode(⟨strict⟩=True)
⟨bool⟩ = ⟨m_obj⟩.is_enabled()
⟨bool⟩ = ⟨m_obj⟩.is_global()

⟨t_obj⟩ = LookupTable(⟨encoded_name⟩, ⟨description⟩,
⟨encoded_die_roll_spec⟩, ⟨seq⟩=None, ⟨group⟩=None, ⟨flags⟩=None,
⟨client_data⟩=None, ⟨table⟩=None)
⟨t_obj⟩ = LookupTable.New(⟨name⟩, ⟨description⟩, ⟨die_roll_spec⟩,
⟨seq⟩=None, ⟨group⟩=None, ⟨flags⟩=None, ⟨client_data⟩=None,
⟨table⟩=None)
⟨pre⟩ = ⟨t_obj⟩.encode(⟨strict⟩=True)
⟨bool⟩ = ⟨t_obj⟩.uses_markup()
⟨str⟩ = ⟨t_obj⟩.lookup_value(⟨n⟩)

main(⟨argv⟩=None, ⟨verbose⟩=False)
```

DESCRIPTION

This module provides a generic die-rolling interface.

There is a low-level `Dice` object which models a set of dice and handles most general random number generation TTRPG games require. See below for a higher-level `DieRoller` interface which is better suited for most applications.

A `Dice` object may be constructed using discrete parameters which together describe a set of dice to be rolled:

⟨*qty*⟩
: The number of dice to roll. Their values will be added together.

⟨*sides*⟩
: Number of sides each die has.

⟨*bonus*⟩
: Additional value to add to the sum of all dice rolled.

⟨*diebonus*⟩
: Additional value added to each die individually.

⟨*div*⟩
: If specified, each die result is divided by ⟨*div*⟩ (fractions truncated).

⟨*factor*⟩
: If specified, this value is multiplied to the final result.

Alternatively, a single string ⟨*description*⟩ may be passed which describes a more complex range of die roll arrangements, as described in `gma-dice-syntax`(7), except for the permutations, global options, and title string.

Regardless of the method used to specify the dice represented by the `Dice` object, it will use a default random number generator such as `random.Random` or the function passed to the `Dice` constructor in the ⟨*generator*⟩ parameter. This function must have a `randint()` method with the same semantics as the `random.Random` generator.

Once the `Dice` object is thus constructed, the following methods may be called on it to roll and examine the dice:

`roll(`⟨*confirm*⟩`=False, `⟨*threat*⟩`=None, `⟨*bonus*⟩`=None)`
: Roll the dice which this `Dice` instance represents. The result is returned as an integer value. Each time this is called, the dice are rerolled to get a new result. The `Dice` object's internal state reflects the last call to this method.

  If the ⟨*confirm*⟩ parameter is `True`, then this roll is for the purpose of confirming a critical roll based on the previous roll, which must have involved only a single die. If the previous roll was the maximum value of the die before other modifiers are applied (or was at least the value of ⟨*threat*⟩ if that was supplied), then another roll is performed and the new result returned. If ⟨*bonus*⟩ is supplied, it is added to the result as well. If there was no previous roll, an exception is raised.

  If the previous roll did not meet the qualifications above (e.g., it was not a natural 20 on a d20), then no roll is performed. In this case, `None` is returned.

`value()`
: Return the value of the last `roll()` of the dice. This is the same value originally returned by `roll()` or `None` if there was no previous roll.

`maxvalue()`
: Return the maximum possible roll that the dice could produce. This counts as a roll of the dice, so subsequent calls to `value()`, `describe_roll()`, and `structured_describe_roll()` will return or describe the maximized value.

`description()`
: This returns a string that describes, in plain text, the dice rolling expression the object was constructed to represent. It is close to, but not identical to, the format of the ⟨*description*⟩ parameter to the constructor.

`describe_roll()`
: This returns a string that describes, in plain text, the results of the most recent `roll()`.

`structured_describe_roll(sf=`⟨*None*⟩`)`
: This returns a detailed accounting of the results of the most recent `roll()`. If `sf` is provided, it is a sequence of two values. If the die-roll involved a single die and that die came up as a natural 1, the reported description includes `('fail',`⟨*sf*⟩`[1])` to indicate automatic failure. If the die was naturally the maximum value possible for the die type, `('success',`⟨*sf*⟩`[0])` to show automatic success. The return value is a ⟨*structured_results*⟩ list. Each element of the list describes a discrete part of the results as a tuple of (⟨*type*⟩, ⟨*string*⟩) where ⟨*type*⟩ is a word that describes what the ⟨*string*⟩ value means. It can be used to select an appropriate formatting for the ⟨*string*⟩. The possible values for ⟨*type*⟩ include "`begingroup`", "`best`", "`bonus`", "`constant`", "`diebonus`", "`diespec`", "`discarded`", "`endgroup`", "`fail`", "`label`", "`max`", "`maximized`", "`maxroll`", "`min`", "`moddelim`", "`operator`", "`result`", "`roll`", "`separator`", "`success`", and "`worst`". If no roll has yet been made, it returns `None` instead.

Higher-Level Interface

A more comprehensive die-rolling interface is provided by the `DieRoller` class. Once instantiated, a new random number is generated by calling the following method:

`do_roll(`⟨*dice*⟩`=)`

If specified, the ⟨*dice*⟩ parameter is a string that describes the die-rolling expression as described in `gma-dice-syntax`(7).

If the ⟨*dice*⟩ string is empty, `do_roll()` re-rolls the previously-specified die-roll expression. If no ⟨*dice*⟩ string was ever given, the `DieRoller` defaults to rolling 1d20.

This method returns a tuple of values (⟨*title*⟩, ⟨*result_set*⟩) representing the results of rolling the dice. The ⟨*title*⟩ is the title specified in the ⟨*dice*⟩ string, or `None` if one was not given. The ⟨*result_set*⟩ is a list of results, one for each roll of the dice that was performed. Each element of ⟨*result_set*⟩ is a tuple (⟨*result*⟩, ⟨*structured_results*⟩) which give the integer result of that die roll, and a structured description of it as described for the `Dice.structured_describe_roll()` method, with the additional ⟨*type*⟩

values of "critlabel", "critspec", "dc", "exceeded", "fail", "fullmax", "iteration", "met", "repeat", "short", "sf", "success", and "until".

`to_text(`⟨*structured_results*⟩`,`⟨ *label*⟩`=None,`⟨ *formatting*⟩`=None)`
: This renders the ⟨*structured_results*⟩ list returned by the `Dice.structured_describe_roll()` and `DieRoller.do_roll()` methods into a string according to the formatting rules given by ⟨*formatting*⟩ (or simply producing a simple plain text version if no ⟨*formatting*⟩ value is given). If ⟨*label*⟩ is given, it is the overall label (called title elsewhere in this document) for the die roll.

  The ⟨*formatting*⟩ dictionary maps ⟨*type*⟩ names as returned by the above-named methods to format strings such as would be accepted by the `str.format()` method, with the corresponding ⟨*value*⟩ provided as the only positional parameter. Thus, the value in ⟨*formatting*⟩`[`⟨*type*⟩`]` is a string which describes a ⟨*value*⟩ of that ⟨*type*⟩, with a "`{}`" or "`{0}`" at the place(s) in the string where the actual ⟨*value*⟩ string should go (see `str.format()` for details).

Interactive Usage

The module's `main()` function may be called to enter an interactive loop. The user is prompted for a die roll, that is fed into `DieRoller.do_roll()`, and the process repeats until the user enters an EOF, or one of the strings "`exit`", "`quit`", or "`q`". If command-line parameters are found in ⟨*argv*⟩, it rolls each dice string in that list instead. If ⟨*verbose*⟩ is `True`, then the raw data structures returned by the underlying API functions are printed as well.

PRESET FILES

If you wish to save a set of prepared die-roll specifications for later use, store them as a collection of `DieRollPreset` objects in a `DieRollPresetCollection` object. The latter behaves just like a regular `list` value, but also has the following additional methods:

`set_timestamp(`⟨*timestamp*⟩`=0)`
: This sets the creation or modification time and date for the collection. This will be written to the file, and may be read in the `DieRollPresetCollection` attribute `timestamp` after loading the file back in. If passed a value of 0 (the default), the current time of day will be set into the `timestamp` and `date_time` automatically.

`save_preset_file(`⟨*f*⟩`)`
: Given an open file-like object ⟨*f*⟩, this writes the contents of the collection to that file.

`write_preset_file(`⟨*path*⟩`)`
: Like `save_preset_file`, but takes the pathname of the target file, which will be opened and written to, then closed.

`load_preset_file(`⟨*f*⟩`)`
: Given an open file-like object ⟨*f*⟩, this reads previously-saved values into the collection. If the file is an older format, it will be read in and converted to the new data structure, but if written back to disk will be written in the current format.

`read_preset_file(`⟨*path*⟩`)`
: Like `load_preset_file`, but reads from the given file pathname, which is automatically opened, read from, and then closed.

The collection also has the following attributes:

`timestamp`
: The integer timestamp giving the time the preset data was last modified.

`date_time`
: A string representation of the `timestamp` value, in any form. This is only for human readers. Software should always use `timestamp` instead.

`comment`
: An arbitrary comment you wish to associate with the die-roll preset collection.

Each element of a `DieRollPresetCollection` is an object of type `DieRollPreset`. To construct one, call the constructor with parameters:

⟨*name*⟩
: A user-defined name for this die-roll specification. Programs will generally display the presets after sorting them on this field, but will not display anything before the first vertical bar (`|`) character in the name. This allows arbitrary sort orders to be maintained. The ⟨*name*⟩ must be unique among the presets in the collection.

⟨*description*⟩
: An arbitrary description explaining what the preset is for.

⟨*die_roll_spec*⟩
: The specification of how to make the die roll as accepted by the `DieRoller.do_roll` method.

A `DieRollPreset` has the methods

`from_tuple((`⟨*name*⟩`, `⟨*description*⟩`, `⟨*die_roll_spec*⟩`))`
: Set the internal values from a tuple of discrete strings. This is provided for backward compatibility with previous implementations.

`to_tuple()`
: Returns a tuple of three strings (as the inverse of the `from_tuple` method).

`to_dict()`
: Returns a dictionary form of the `DieRollPreset` values.

COMPLEX PRESET OBJECTS

Die-Roll Presets, in their simplest concept, are simply a tuple of strings (⟨*name*⟩, ⟨*description*⟩, ⟨*expression*⟩) which are stored server-side and made available to clients to facilitate the convenient re-use of common die rolls. However, in practice they have been used to represent a number of more complex objects, the details of which have been encoded into these strings in ways which clients are free to interpret. The server's only involvement is to store and retrieve them for the clients. These are stored in the `DieRollPreset` objects described above, which are the objects sent to and received from the server via the API.

To make it more coinvenient to work with the more complex data objects represented by (i.e., encoded within) the preset tuples, we present in this Python module a number of more specific object classes and a factory function which will allow you to work directly with the higher-level object without worrying about how it's represented or encoded for storage.

Given a `DieRollPreset` object obtained from the server as described above, pass it to the `get_preset_details()` factory function to extract the higher-level detailed object from it. This returns a new object which will be one of the types described below.

If you make changes to the returned object and wish to save it back to the server, you can put it back into an encoded `DieRollPreset` object by calling its `encode()` method, which returns a new `DieRollPreset` object with its current values encoded into it, suitable for passing along to the server via the API.

Collections of `DieRollObjects` may be managed together in a `DieRollObjectCollection` object (which is a subclass of a Python `list` type), just as `DieRollPresetCollection` is a type that represents a list of `DieRollPreset` values. Both of these collection types have `save_preset_file()`, `write_preset_file()`, `load_preset_file()`, and `read_preset_file()` methods to read and write presets to disk files. `DieRollObjectCollection` methods for these operations allow you to read and write them directly, encoding and decoding them as needed without requiring you to explicitly convert them to and from the underlying `DieRollPreset` type yourself.

At the collection level, you can convert an entre list of `DieRollPreset` values to `DieRollObject` types by calling the `get_details()` method of the `DieRollPresetCollection`. Conversely the `DieRollObjectCollection`'s `encode()` method will re-encode all of its elements back into `DieRollPreset` values and return them as a new `DieRollPresetCollection` list.

Given a `DieRollObjectCollection`, calling its `resequence()` method will sort its elements in-place by groups, and update all of their `seq` attributes appropriately so they will display to the user in the new sorted order.

The following sections describe the subclasses of `DieRollObject` which represent each of the various specific types of complex die-roll preset object.

Simple Die-Roll Presets: SimplePreset (DieRollObject)

In the simplest case, a basic preset is represented by a `SimplePreset` object. This is usually created for you by the `get_preset_details()` factory function which takes an existing `DieRollPreset` value on input and creates the `SimplePreset` for you based on it

⟨*s_obj*⟩ = `get_preset_details(`⟨*preset*⟩`)`

but you may also create one manually by invoking the `SimplePreset` constructor with three string parameters

⟨*s_obj*⟩ = `SimplePreset(`⟨*name*⟩`,` ⟨*description*⟩`,` ⟨*die_roll_spec*⟩`,` ⟨*seq*⟩`=None,` ⟨*group*⟩`=None,` ⟨*flags*⟩`=None,` ⟨*client_data*⟩`=None)`

These are the same as the three strings stored server-side by the server. Simple presets offer a few features over and above the simple strong storage of die roll specification strings described above. They also allow for encoding hidden sequencing and grouping information as well as flags and client data in the ⟨*name*⟩ field, which allow presets to be ordered in an arbitrary sequence for the user when sorted by the sequence value, although that is not actually displayed to the user. The groups allow presets to be organized into meaningful groups (or even hierarchies of groups) as needed.

If you want to specify these directly to the constructor, overriding any which may be encoded into the ⟨*name*⟩ field, pass them to the named parameters ⟨*seq*⟩ (for the sort sequence, which MAY be almost any string value, but MUST NOT contain semicolons or vertical bars; we RECOMMEND just using decimal digits here) and ⟨*group*⟩ (which is a list of strings, which should be simple alphanumeric values themselves, indicating the group level(s) for where you want this preset to be organized in your die roller).

The `SimplePreset` object has the following attributes:

`type`
: This is always `preset` and should not be changed.

`name`
: The display name of the preset as seen by the user (the first tuple string, stripped of hidden encoded data fields).

`description`
: The description of the preset as set by the user (the second tuple string on the server).

`die_roll_spec`
: The die-roll specification expression (what will actually be rolled) as set by the user (the third tuple string).

`client_data`
: Arbitrary information the client wishes to store. This MUST NOT contain semicolons or vertical bars.

`flags`
: A set of flags which specify options in effect for this preset. The only currently recognized flag for this type is:

  `b`
  : By default, send the die roll blindly to the GM only.

`group`
: A list of the group(s) containing this preset, with the first element being the top-level group, and so forth down for each nested sub-group (if any), or `None` if the preset does not belong to any group.

`seq`
: The display sequence set by the user for this preset. This is an arbitrary string but in practice we recommend using decimal digit strings for these with leading zeroes so they sort in ASCII order but don't interfere with other text encoded into the field. This may be `None` if there was no auxiliary data encoded into the first tuple string at all, or the empty string if there was, but it included no sequence data.

The following method functions are available:

`encode()`
: This method creates and returns a new `DieRollPreset` object which represents this simple preset data, suitable for passing up to API functions which only understand those. The `SimplePreset` object itself is unaffected.

`should_roll_blind_to_GM()`
: This returns `True` if the `b` flag is set.

Dictionary Presets: DictPreset (DieRollObject)

Clients can store arbitrarily complex preset data into a server preset via a `DictPreset` object, which allows its `die_roll_spec` attribute to be a dictionary of values instead of a single string value.

If a `DieRollPreset` object representing a dictionary preset is passed to the `get_preset_details()` function as in ⟨*d_obj*⟩ = `get_preset_details(`⟨*preset*⟩`)`

it will return the details about this preset data in the form of a `DictPreset` object. You can also create one directly by calling its constructor with the three encoded tuple strings: ⟨*d_obj*⟩ = `DictPreset(`⟨*encoded_name*⟩`,` ⟨*description*⟩`,` ⟨*die_roll_spec*⟩`,` ⟨*seq*⟩`=None,` ⟨*group*⟩`=None,` ⟨*flags*⟩`=None,` ⟨*client_data*⟩`=None,` ⟨*table*⟩`=None)`

Note that you can additionally pass named parameters following the three tuple strings to override the sequence, group, flags, client data, and

table values to replace what was encoded into the tuples with new values if desired.

If you want to create a new `DictPreset` from scratch without having an already-encoded tuple string to start with, call the `New()` class method instead, which allows you to specify the display name you want to call this preset for the ⟨*name*⟩ parameter, instead of the string which has the group, sequence, and other information packed into it as well: ⟨*d_obj*⟩ = `DictPreset.New(`⟨*name*⟩`,` ⟨*description*⟩`,` ⟨*die_roll_spec*⟩`,` ⟨*seq*⟩`=None,` ⟨*group*⟩`=None,` ⟨*flags*⟩`=None,` ⟨*client_data*⟩`=None,` ⟨*table*⟩`=None)`

`DictPreset` objects may, like `SimplePreset` objects, include nested groups and sequence information. `DictPreset` objects have the following attributes:

`type`
: This is always `dict` and should not be changed.

`name`
: The display name of the preset as seen by the user (the first tuple string, stripped of hidden encoded data fields).

`client_data`
: Arbitrary data string which may be of use to other programs reading the preset. This field MUST NOT contain semicolons or vertical bars. To be safest, it should probably be encoded as base 64 if it will contain totally arbitrary data to avoid issues with surrounding data structures.

`description`
: The description of the preset as set by the user (the second tuple string on the server).

`die_roll_spec`
: The value of the preset as set by the user (as encoded in the third tuple string), as a Python dictionary value.

`group`
: A list of the group(s) containing this preset, with the first element being the top-level group, and so forth down for each nested sub-group (if any), or `None` if the preset does not belong to any group.

`seq`
: The display sequence set by the user for this preset. This is an arbitrary string but in practice we recommend using decimal digit strings for these with leading zeroes so they sort in ASCII order but don't interfere with other text encoded into the field. This may be `None` if there was no auxiliary data encoded into the first tuple string at all, or the empty string if there was, but it included no sequence data.

The following method function is available:

`encode()`
: This method creates and returns a new `DieRollPreset` object which represents this preset data, suitable for passing up to `API` functions which only understand those. The `DictPreset` object itself is unaffected.

Modifiers and Variables: DieRollModifier (DieRollObject)

Some objects stored in the preset data aren't actually rolls but are modifiers to be added to other die-roll expressions either by appending them to the end, or assigning them to variable names which can be substituted into expressions anywhere desired by the user.

If a `DieRollPreset` object representing such a modifier is passed to the `get_preset_details()` factory function

⟨*m_obj*⟩ = `get_preset_details(`⟨*preset*⟩`)`

it will return the details about this modifier in the form of a `DieRollModifier` object. You can also create one directly from the three tuple strings by calling its constructor

⟨*m_obj*⟩ = `DieRollModifier(`⟨*encoded_name*⟩`,` ⟨*description*⟩`,` ⟨*die_roll_spec*⟩`,` ⟨*seq*⟩`=None,` ⟨*group*⟩`=None,` ⟨*varname*⟩`=None,` ⟨*flags*⟩`=None,` ⟨*client_data*⟩`=None)`

Specifying any of the named parameters following the required three encoded tuple strings will override the corresponding values that were decoded out of the strings.

If you wish to create a new `DieRollModifier` from scratch without starting from an encoded tuple, call the `New()` class method instead. In this case, the ⟨*namd*⟩ parameter is not encoded with the other data fields. It is just the display name visible to the user.

⟨*m_obj*⟩ = `DieRollModifier.New(`⟨*name*⟩`,` ⟨*description*⟩`,` ⟨*die_roll_spec*⟩`,` ⟨*seq*⟩`=None,` ⟨*group*⟩`=None,` ⟨*varname*⟩`=None,` ⟨*flags*⟩`=None,` ⟨*client_data*⟩`=None)`

`DieRollModifier` objects include the same features as described above for `SimplePreset` objects, including sequence and group encodings. In addition, we can assign a variable name to the modifier which the client can make available for the user to include in die-roll expressions, and a number of flags which can be set to control how the client will apply the modifier. We can also store arbitrary client data into the modifier for whatever purposes the client needs to use that for.

The `DieRollModifier` object has the following attributes:

`type`
: This is always `modifier` and should not be changed.

`name`
: The display name of the modifier as seen by the user (the first tuple string, stripped of hidden encoded data fields). Note that this is not the variable name (see below).

`description`
: The description of the modifier as set by the user (the second tuple string on the server).

`die_roll_spec`
: The value of the modifier expression as set by the user (the third tuple string).

`group`
: A list of the group(s) containing this modifier, with the first element being the top-level group, and so forth down for each nested sub-group (if any), or `None` if the modifier does not belong to any group.

`seq`
: The display sequence set by the user for this modifier. This is an arbitrary string but in practice we recommend using decimal digit strings for these with leading zeroes so they sort in ASCII order but don't interfere with other text encoded into the field. This may be `None` if there was no auxiliary data encoded into the first tuple string at all, or the empty string if there was, but it included no sequence data.

`flags`
: A set containing flag values which are in effect for the modifier. These are arbitrary and entirely up to the client to manage and interpret. Currently the following flag values are allowed:

  `'e'` The modifier is enabled (will be added to all die rolls, or if a variable name is assigned, that name is available for use in die rolls).

  `'g'` The modifier is not just appended as-is to the end of the die-roll expression. When this flag is in effect, we throw a pair of `(...)` brackets around the expression first so the modifier applies to the result of the whole expression so far at that point.

`client_data`
: Arbitrary data the client finds useful to store here which it will need to have on hand when using the modifier later. This data MUST NOT contain semicolons or vertical bars. This is `None` if no client data was defined.

`varname`
: The name of the variable this modifier is available to the user as, if any, or `None` if no variable name was defined for it.

The following method functions are available:

`encode(`⟨*strict*⟩`=True)`
: This method creates and returns a new `DieRollPreset` object which represents this modifier data, suitable for passing up to API functions which only understand those. The `DieRollModifier` object itself is unaffected.

  If the ⟨*strict*⟩ parameter is `True`, strict conformity to the spec is enforced. For example, only the expected flag values will be preserved. If `False`, then all flag values present in the `flags` attribute will be saved, but note that flags are required to be single characters so only the first character of each flag string in `flags` will be kept.

`is_enabled()`
: Returns `True` if the `'e'` flag is set in this object's `flags` attribute.

`is_global()`
: Returns `True` if the `'g'` flag is set in this object's `flags` attribute.

Lookup Tables: LookupTable (DieRollObject)

In many fantasy role-playing games there are cases where you want to roll dice and look up the random result on a table to see what the outcome means. For convenience, especially when making many such rolls in bulk quantities, it may be convenient to automate these table lookups. To help with this, die roll presets may also include such lookup tables as one of the forms of complex objects stored there.

If a `DieRollPreset` object representing such a table is passed to the `get_preset_details()` factory function

⟨*t_obj*⟩ = `get_preset_details(`⟨*preset*⟩`)`

it will return the details about this table in the form of a `LookupTable` object. You can also create one directly from the three tuple strings by calling its constructor

⟨*t_obj*⟩ = `LookupTable(`⟨*encoded_name*⟩, ⟨*description*⟩, ⟨*encoded_die_roll_spec*⟩, ⟨*seq*⟩`=None`, ⟨*group*⟩`=None`, ⟨*flags*⟩`=None`, ⟨*client_data*⟩`=None`, ⟨*table*⟩`=None)`

Adding any of the named parameters will override the corresponding values that were decoded out of the tuple strings with new values. Alternatively, you can create a wholly new `LookupTable` object without starting from an already-encoded set of strings by calling the `New()` class method:
⟨*t_obj*⟩ = `LookupTable.New(`⟨*name*⟩, ⟨*description*⟩, ⟨*die_roll_spec*⟩, ⟨*seq*⟩`=None`, ⟨*group*⟩`=None`, ⟨*flags*⟩`=None`, ⟨*client_data*⟩`=None`, ⟨*table*⟩`=None)`

`LookupTable` objects also include the same basic features as described above for `SimplePreset` objects, including sequence and group encodings. Beyond that, we can set flags and define the table itself as well as storing client data. The die-roll expression is assumed to be specifically what should be rolled when consulting the table.

The `LookupTable` object has the following attributes:

`type`
: This is always `table` and should not be changed.

`name`
: The display name of the table as seen by the user (the first tuple string, stripped of hidden encoded data fields). Clients typically add prefix characters such as # or ## in front of this name when rolling on the table, but the name as given here does not include those.

`description`
: The description of the table as set by the user (the second tuple string on the server).

`die_roll_spec`
: The die (or die-roll expression) to roll when rolling on the table, derived from the data encoded into the third tuple string.

`group`
: A list of the group(s) containing this table, with the first element being the top-level group, and so forth down for each nested sub-group (if any), or `None` if the table does not belong to any group.

`seq`
: The display sequence set by the user for this table. This is an arbitrary string but in practice we recommend using decimal digit strings for these with leading zeroes so they sort in ASCII order but don't interfere with other text encoded into the field. This may be `None` if there was no auxiliary data encoded into the first tuple string at all, or the empty string if there was, but it included no sequence data.

`flags`
: A set containing flag values which are in effect for the table. These are arbitrary and entirely up to the client to manage and interpret. Currently the following flag values are allowed:

  `'b'` Make die rolls blindly to the GM only by default.

  `'m'` GMA markup formatting codes may be present in the table's text, so clients should try to resolve that and render the text correctly when displaying it to the user.

`client_data`
: Arbitrary data the client finds useful to store here which it will need to have on hand when using the table later. This value MUST NOT contain any semicolons or vertical bars. This is `None` if no client data was defined.

`delimiter`
: The delimiter character used when encoding the table definition into the preset for storage into the encoded `DieRollPreset` object. This can

be any character but must not appear anywhere in the table data. If you change the table data, this will need to continue to be a character that does not appear in that data. This can be accomplished by calling the `seek_delimiter()` method, which is automatically called for you by the `encode()` method.

`table`
: The lookup table itself, as a list of ⟨*N*⟩+1 pairs of values

  `[`⟨*n0*⟩`,` ⟨*t0*⟩`,` ⟨*n1*⟩`,` ⟨*t1*⟩`, ...` ⟨*n(N–1)*⟩`,` ⟨*t(N–1)*⟩`, '*',` ⟨*tN*⟩`]`

  where ⟨*N*⟩≥0. The values of ⟨*n0*⟩, ⟨*n1*⟩, etc. must appear in ascending order and must be integer values except for the final one, which must be the string `'*'`. In each case, the corresponding string ⟨*t*⟩ is to be used for a die-roll result less than or equal to the value ⟨*n*⟩, with the final case (`'*'`) being a catch-all default case that matches any value not yet matched by any previous row in the table. For example, the lookup table

1–5	Do one thing
6	Do nothing
7–8	Make something up
9+	Run away

  would be represented in the `table` attribute as the list value

```
[5, 'Do one thing', 6, 'Do nothing', 8, 'Make something up', '*',
'Run away']
```

The following method functions are available:

`encode(`⟨*strict*⟩`=True)`
: This method creates and returns a new `DieRollPreset` object which represents this table data, suitable for passing up to `API` functions which only understand those. The `LookupTable` object itself is unaffected, unless the `delimiter` attribute needs to be changed to properly encode the table's data. An exception is raised if the object cannot correctly be encoded into a `DieRollPreset` object.

  If the ⟨*strict*⟩ parameter is `True`, strict conformity to the spec is enforced. For example, only the expected flag values will be preserved. If `False`, then all flag values present in the `flags` attribute will be saved, but note that flags are required to be single characters so only the first character of each flag string in `flags` will be kept.

`uses_markup()`
: Returns `True` if the `'m'` flag is set in this object's `flags` attribute.

`lookup_value(`⟨*n*⟩`)`
: When passed an integer value ⟨*n*⟩, this looks up that value on the table and returns the corresponding entry in the table. It does this by starting at the top of the table and iterating until it finds an entry whose associated value is greater than or equal to ⟨*n*⟩, or until it reaches the special case `*`.

`seek_delimiter(`⟨*desired_delimiter*⟩`=None)`
: Attempt to find a suitable delimiter character to use for encoding the `LookupTable` back into the `DieRollPreset` object as it will be stored on the server. Normally this method will look through an internal set of potential delimiter characters to try, but if you wish, you can pass a string parameter containing one or more delimiter characters of your own preference which it will attempt first. When a suitable delimiter is found (one that does not conflict with any of the existing data in the `LookupTable` object as it currently stands), the object's `delimiter` attribute will be changed accordingly. If one cannot be found, an exception is raised. This function is called automatically by the `encode()` method, so it is not strictly necessary for you to call it yourself.

`should_roll_blind_to_GM()`
: This returns `True` if the `b` flag is set.

STRUCTURED RESULTS

Some methods return a structured result list to describe the outcome of the die roll. This is a human-readable text description of the rolls made, broken into pieces to facilitate easier formatting. Each element of the structured results list is a (⟨*type*⟩, ⟨*value*⟩) tuple. The ⟨*type*⟩ explains how to format the corresponding ⟨*value*⟩, and may be one of the following, although you should accept ⟨*type*⟩ names not in this list as well:

`begingroup`
: Begins a new sub-expression group.

`best`
: The specifier to reroll and take the best of the dice rolled. The value is just the number of die rolls to perform.

`bonus`
: Overall bonus applied to the entire roll but not included in the expression (e.g., the bonus added to confirm critical rolls).

`constant`
: A constant value being added (or whatever) to the overall total.

`critlabel`
: Label automatically given to confirmation rolls.

`critspec`
: Specifier for confirming critical rolls.

`cumulative`
: The cumulative total so far (if using the `|total` option).

`dc`
: The difficulty class for this roll.

`diebonus`
: Bonus value added to each die in an individual diespec (not the overall total).

`diespec`
: The specification of a single die roll component, such as "`12d6`".

`discarded`
: Results of a die roll that was made but discarded due to another rule, as a comma-separated list of numbers that were rolled.

`endgroup`
: Ends a sub-expression group.

`error`
: An error was encountered when trying to satisfy the die-roll request. The value has the error message string.

`exceeded`
: The amount by which the roll exceeded the `DC`.

`fail`
: The result of a rolled percentile check roll that indicated failure.

`fullmax`
: The option "`maximized`" which requests all dice to be maximized.

`iteration`
: The roll number of the current iteration of rolls for repeating sets of rolls.

`label`
: Arbitrary text label attached to a die expression (e.g., the "`fire`" in "`1d6 fire`").

`max`
: The maximum value allowed for the die roll.

`maximized`
: The prefix ">" which requests a maximized first die in the roll.

`maxroll`
: Same as `roll`, but all die rolls were forced to be maximized.

`met`
: An indicator that the DC was met exactly.

`min`
: The minimum value for the die roll.

`moddelim`
: Delimiters which separate global modifiers from the other parts of the expression.

`notice`
: A general comment the user should know about this die roll.

`operator`
: One of the operators (“+”, “-”, etc.) which separates each die roll expression.

`repeat`
: The number of times the roll is being repeated.

`result`
: The full result of the die roll.

`roll`
: The results of rolling a single diespec, as a comma-separated list of numbers representing the side that rolled for each die in the diespec.

`separator`
: Separators such as the “=” between the title and the rest of the expression.

`short`
: The amount by which the roll fell short of the DC.

`success`
: The result of a rolled percentile check roll that indicated success.

`total`
: The value we are aiming for as a cumulative total of all rolls.

`until`
: The value we are rolling repeatedly until we reach.

`worst`
: The specifier to reroll and take the worst of the dice rolled. The value is the number of times to try rolling.

For example, calling `DieRoller.do_roll('1d20+3 | min 5 | c')` might result in an initial natural 20 (giving an overall result of 23). But since that was a natural 20, a second confirmation roll is made (due to the "`c`" at the end of the expression), with a result of 13. In this case, the value returned from `do_roll()` is:

```
(None, [
    (23, [
        ('result', '23'),
        ('success', 'HIT'),
        ('separator', '='),
        ('diespec', '1d20'),
        ('roll', '20'),
        ('operator', '+'),
        ('constant', '3'),
        ('moddelim', '|'),
        ('min', '5'),
        ('moddelim', '|'),
        ('critspec', 'c')
    ]),
    (13, [
        ('critlabel', 'Confirm:'),
        ('result', '13'),
        ('separator', '='),
        ('diespec', '1d20'),
        ('roll', '10'),
        ('operator', '+'),
        ('constant', '3'),
        ('moddelim', '|'),
        ('min', '5')
    ])
])
```

Likewise, calling `DieRoller.do_roll('Attack=d20+{17/12}+3')` might return:

```
('Attack', [
    (38, [
        ('result', '38'),
        ('separator', '='),
        ('diespec', '1d20'),
        ('roll', '18'),
```

```
        ('operator', '+'),
        ('constant', '17'),
        ('operator', '+'),
        ('constant', '3')
    ]),
    (35, [
        ('result', '35'),
        ('separator', '='),
        ('diespec', '1d20'),
        ('roll', '20'),
        ('operator', '+'),
        ('constant', '12'),
        ('operator', '+'),
        ('constant', '3')
    ])
]
```

DIAGNOSTICS

These functions may raise the following exceptions in addition to the normal set that may occur otherwise:

`InvalidDiceDescription`
: The specification of the dice to roll was not understandable as written.

`InvalidConfirmUsage`
: A confirmation was requested for a die roll specification that could not apply to.

SEE ALSO

`gma-roll`(6), `gma-go-roll`(6), `gma-dice-syntax`(7).

AUTHOR

Steve Willoughby / steve@madscience.zone.

BUGS

As a compatibility measure, a `DieRollPresetCollection` may also contain elements which are 3-element tuples or lists. This may change in favor of requiring them to all be `DieRollPreset` objects in the future.

gma-dicepresetcollection(3)

See gma-mapservice(3), p. 284.

gma-dieroller(3)

See gma-dice(3), p. 246.

gma-eventcollection(3)

See gma-mapservice(3), p. 284.

gma-feet_to_centimeters(3)

See gma-weightsandmeasures(3), p. 351.

gma-from_roman(3)

See gma-romannumerals(3), p. 318.

gma-grams_to_pounds(3)

See gma-weightsandmeasures(3), p. 351.

gma-hexdump(3)

NAME

hexdump – Utility to create hex dumps of data

SYNOPSIS

```
import MadScienceZone.Common.hexdump

⟨string⟩ = hexdump(⟨data⟩, ⟨addr⟩=0, ⟨width⟩=16, ⟨word⟩=1,
⟨ASCII⟩=True)
```

DESCRIPTION

This module provides a `hexdump()` function which accepts a ⟨*data*⟩ string and emits a representation of that data as a hexadecimal dump. This output string may have embedded newlines and could possibly be quite large, depending on the size of ⟨*data*⟩.

The optional parameters to `hexdump` are:

⟨*addr*⟩
: The hex dump includes addresses in the left column of output which indicates the position in the data for each line of hex values. By default, these addresses begin with 0, but this parameter may be given to provide any starting value for them.

⟨*width*⟩
: The number of bytes worth of data to format onto each line of output.

⟨*word*⟩
: The number of bytes to show in each value.

⟨*ASCII*⟩
: If true, include the ASCII values of the bytes in the right column of output. Unprintable characters are represented by a dot (“.”).

For example, calling

```
hexdump('Spam, spam, spam,\neggs, spam!')
```

returns a string with this value:

```
00000000:  53 70 61 6D 2C 20 73 70 61 6D 2C 20 73 70 61 6D  |Spam, spam, spam|
00000010:  2C 0A 65 67 67 73 2C 20 73 70 61 6D 21           |,.eggs, spam!   |
```

SEE ALSO

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

gma-htmlformatter(3)

See gma-markuptext(3), p. 298.

gma-inches_to_centimeters(3)

See gma-weightsandmeasures(3), p. 351.

gma-init_application(3)

See gma-preferences(3), p. 310.

gma-load_map_file(3)

See gma-mapobject(3), p. 274.

gma-load_map_meta_data(3)

See gma-mapobject(3), p. 274.

gma-lockfile(3)

NAME

LockFile – Simple lock file manager

SYNOPSIS

```
import MadScienceZone.Common.LockFile
⟨lf⟩ = LockFile(⟨filename⟩, ⟨timeout⟩=60, ⟨force⟩=0,
⟨mode⟩=0o644, ⟨comment⟩=None, ⟨verbose⟩=False)
```

DESCRIPTION

A `LockFile` object instance manages a single lockfile in the filesystem. Once created by calling the constructor, the file exists, but will be removed if possible when this object is destroyed or the unlock() method is called on it.

The constructor invocation syntax is:

`LockFile(`⟨*filename*⟩`,` [⟨*timeout*⟩`,`] [⟨*force*⟩`,`] [⟨*mode*⟩`,`] [⟨*comment*`,`⟩] [⟨*verbose*⟩])

where:

⟨*filename*⟩
: is the pathname to be created as the lockfile.

⟨*timeout*⟩
: is the time in seconds to allow an existing lock to clear before giving up. [default: 60 sec]

⟨*force*⟩
: if ⟨*force*⟩>0, force deletion of lockfile if timeout expired and lockfile is older than ⟨*force*⟩ seconds. [default: 0, i.e., do not force]

⟨*mode*⟩
: permission bits to use for creating file [default: 0644, i.e., `rw-r-r-`]

⟨*comment*⟩
: text to go into lockfile in case that helps others encountering it while it exists. [default: None]

The constructor parameters are retained as object attributes with the same names.

Not being able to set the file permission mode is ⟨*not*⟩ considered an error.

If successful, a lockfile is created and an object returned. Otherwise, an exception is raised. This exception will be `LockFileBusy` if the reason for the failure was that there is an unexpired lockfile already present.

The lockfile is created in a way that is as atomic as possible to avoid failure due to race conditions. It is believed to be able to reliably create lockfiles atomically even on an NFS volume.

Methods

Given a successfully created `LockFile` object ⟨*lf*⟩, the following methods are available:

⟨*lf*⟩`.unlock()`
: Release the lock by deleting the file from the filesystem. At this point the object is useless and can be discarded.

⟨*lf*⟩`.persist()`
: Normally, the file is automatically deleted when the ⟨*lf*⟩ object is destructed (assuming normal object destruction and program termination; if your program stops unexpectedly as on a signal or exception, this is not guaranteed). Calling this function arranges for the file to persist after this object is destroyed. Removing the file from the filesystem is now your responsibility.

INTERACTIVE USE

Running this module as a standalone Python script produces the behavior documented in `gma-lockfile`(1).

SEE ALSO

`gma-lockfile`(1).

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

gma-map_label_to_xy(3)

See gma-mapobject(3), p. 274.

gma-mapcharacter(3)

See gma-socketinterface(3), p. 322.

gma-mapcondition(3)

See gma-socketinterface(3), p. 322.

gma-mapevent(3)

See gma-mapservice(3), p. 284.

gma-mapioclientconnection(3)

See gma-mapservice(3), p. 284.

gma-mapiodispatcher(3)

See gma-mapservice(3), p. 284.

gma-mapobject(3)

NAME

MapObject – Model for mapper elements

SYNOPSIS

```
import MadScienceZone.GMA.Mapper.MapObject

MINIMUM_SUPPORTED_MAP_FILE_FORMAT = 20
MAXIMUM_SUPPORTED_MAP_FILE_FORMAT = 20

⟨o⟩ = MapObject(⟨ID⟩)
⟨o⟩ = MapObject.from_json(⟨dict⟩)
⟨id⟩ = ⟨o⟩.ObjID()
⟨dict⟩ = ⟨o⟩.to_json()

⟨e⟩ = MapElement(⟨ID⟩, ⟨x⟩, ⟨y⟩, ⟨z⟩, ⟨hidden⟩=False,
⟨locked⟩=False, ⟨dash⟩=DashType.SOLID, ⟨width⟩=0, ⟨level⟩=0,
⟨points⟩=None, ⟨line⟩=None, ⟨fill⟩=None, ⟨layer⟩=None,
⟨group⟩=None)
⟨e⟩ = MapElement.from_json(⟨dict⟩)
⟨dict⟩ = ⟨e⟩.to_json()
⟨id⟩ = ⟨e⟩.ObjID()

⟨arc⟩ = ArcElement(⟨ID⟩, ⟨arc_mode⟩=ArcModeType.PIESLICE,
⟨start⟩=0, ⟨extent⟩=0, ...)
⟨arc⟩ = ArcElement.from_json(⟨dict⟩)
⟨dict⟩ = ⟨arc⟩.to_json()
⟨str⟩ = ⟨arc⟩.record_type()

⟨circ⟩ = CircleElement(⟨ID⟩, ...)
⟨circ⟩ = CircleElement.from_json(⟨dict⟩)
⟨dict⟩ = ⟨circ⟩.to_json()
⟨str⟩ = ⟨circ⟩.record_type()

⟨line⟩ = LineElement(⟨ID⟩, ⟨arrow⟩=ArrowType.NONE, ...)
⟨line⟩ = LineElement.from_json(⟨dict⟩)
⟨dict⟩ = ⟨line⟩.to_json()
⟨str⟩ = ⟨line⟩.record_type()
```

```
⟨poly⟩ = PolygonElement(⟨ID⟩, ⟨join⟩=JoinStyle.BEVEL, ⟨spline⟩=0,
...)
⟨poly⟩ = PolygonElement.from_json(⟨dict⟩)
⟨dict⟩ = ⟨poly⟩.to_json()
⟨str⟩ = ⟨poly⟩.record_type()
⟨r⟩ = RectangleElement(⟨ID⟩, ...)
⟨r⟩ = RectangleElement.from_json(⟨dict⟩)
⟨dict⟩ = ⟨r⟩.to_json()
⟨str⟩ = ⟨r⟩.record_type()

⟨a⟩ = SpellAreaOfEffect(⟨ID⟩, ⟨aoe_shape⟩=AoEType.CONE, ...)
⟨a⟩ = SpellAreaOfEffect.from_json(⟨dict⟩)
⟨dict⟩ = ⟨a⟩.to_json()
⟨str⟩ = ⟨a⟩.record_type()

⟨t⟩ = TextElement(⟨ID⟩, ⟨text⟩=, ⟨font⟩=None,
⟨anchor⟩=AnchorDirection.CENTER, ...)
⟨t⟩ = TextElement.from_json(⟨dict⟩)
⟨dict⟩ = ⟨t⟩.to_json()
⟨str⟩ = ⟨t⟩.record_type()

⟨i⟩ = TileElement(⟨ID⟩, ⟨image⟩=None, ⟨bb_height⟩=0,
⟨bb_width⟩=0, ...)
⟨i⟩ = TileElement.from_json(⟨dict⟩)
⟨dict⟩ = ⟨i⟩.to_json()
⟨str⟩ = ⟨i⟩.record_type()

⟨c⟩ = CreatureToken(⟨ID⟩, ⟨killed⟩=False, ⟨dim⟩=False,
⟨hidden⟩=False, ⟨creature_type⟩=CreatureTypeCode.UNKNOWN,
⟨move_mode⟩=MoveModeType.LAND, ⟨reach⟩=0, ⟨skin⟩=0, ⟨elev⟩=0,
⟨gx⟩=0, ⟨gy⟩=0, ⟨name⟩=None, ⟨health⟩=None, ⟨skin_size⟩=None,
⟨color⟩=None, ⟨note⟩=None, ⟨size⟩=None, ⟨area⟩=None,
⟨status_list⟩=None, ⟨aoe⟩=None)
⟨c⟩ = CreatureToken.from_json(⟨dict⟩)
⟨dict⟩ = ⟨c⟩.to_json()
⟨str⟩ = ⟨c⟩.record_type()

⟨m⟩ = MonsterToken(...)
⟨m⟩ = MonsterToken.from_json(⟨dict⟩)
⟨dict⟩ = ⟨m⟩.to_json()
⟨str⟩ = ⟨m⟩.record_type()

⟨p⟩ = PlayerToken(...)
⟨p⟩ = PlayerToken.from_json(⟨dict⟩)
⟨dict⟩ = ⟨p⟩.to_json()
⟨str⟩ = ⟨p⟩.record_type()
```

```
⟨pt⟩ = Coordinates(⟨x⟩, ⟨y⟩)
⟨pt⟩ = Coordinates.from_json(⟨dict⟩)
⟨dict⟩ = ⟨pt⟩.to_json()
⟨h⟩ = CreatureHealth(⟨is_flat_footed⟩=False, ⟨is_stable⟩=False,
⟨max_hp⟩=0, ⟨lethal_damage⟩=0, ⟨non_lethal_damage⟩=0,
⟨con⟩=0, ⟨hp_blur⟩=0, ⟨condition⟩=None)
⟨h⟩ = CreatureHealth.from_json(⟨dict⟩)
⟨dict⟩ = ⟨h⟩.to_json()

⟨f⟩ = FileDefinition(⟨is_local_file⟩=False, ⟨file⟩=None)
⟨f⟩ = FileDefinition.from_json(⟨dict⟩)
⟨dict⟩ = ⟨f⟩.to_json()
⟨str⟩ = ⟨f⟩.record_type()

⟨i⟩ = ImageDefinition(⟨name⟩, ⟨sizes⟩=None)
⟨i⟩ = ImageDefinition.from_json(⟨dict⟩)
⟨dict⟩ = ⟨i⟩.to_json()
⟨str⟩ = ⟨i⟩.record_type()

⟨id⟩ = ImageInstance(⟨is_local_file⟩=False, ⟨zoom⟩=0, ⟨file⟩=None,
⟨image_data⟩=None)
⟨id⟩ = ImageInstance.from_json(⟨dict⟩)
⟨dict⟩ = ⟨id⟩.to_json()

⟨a⟩ = RadiusAoE(⟨radius⟩=0, ⟨color⟩=None)
⟨a⟩ = RadiusAoE.from_json(⟨dict⟩)
⟨dict⟩ = ⟨a⟩.to_json()

⟨font⟩ = TextFont(⟨family⟩, ⟨size⟩,
⟨weight⟩=FontWeightType.NORMAL, ⟨slant⟩=FontSlantType.ROMAN)
⟨font⟩ = TextFont.from_json(⟨dict⟩)
⟨dict⟩ = ⟨font⟩.to_json()

⟨m⟩ = MapMetaData(⟨timestamp⟩=0, ⟨date_time⟩=None,
⟨comment⟩=None, ⟨location⟩=None, ⟨file_version⟩=None)
⟨m⟩ = MapMetaData.from_json(⟨dict⟩)
⟨dict⟩ = ⟨m⟩.to_json()
⟨str⟩ = ⟨m⟩.record_type()

write_map_file(⟨path⟩, ⟨obj_list⟩, ⟨meta⟩)
save_map_file(⟨fileobj⟩, ⟨obj_list⟩, ⟨meta⟩)

⟨obj_list⟩, ⟨meta⟩ = read_map_file(⟨path⟩)
⟨meta⟩ = read_map_meta_data(⟨path⟩)
⟨meta⟩ = load_map_meta_data(⟨fileobj⟩)
⟨obj_list⟩, ⟨meta⟩ = load_map_file(⟨fileobj⟩, ⟨meta_only⟩=False)

⟨label⟩ = xy_to_map_label(⟨x⟩, ⟨y⟩)
⟨x⟩, ⟨y⟩ = map_label_to_xy(⟨label⟩)
```

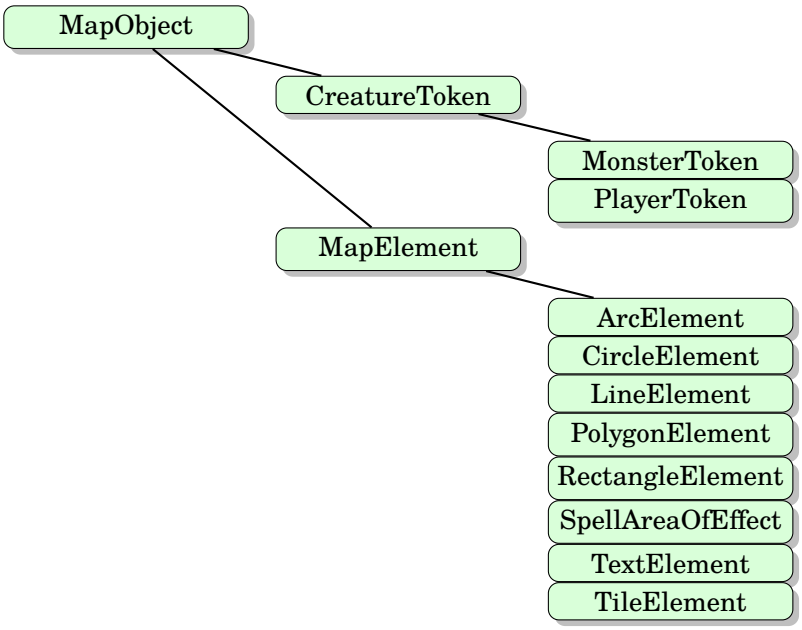


Figure B.1: Map object class hierarchy

DESCRIPTION

The `gma-mapper`(6) program manages a number of "elements" on the display of the battle grid: lines, polygons, image tiles, creature tokens, etc. Programmatically, these may each be represented by a `MapObject` instance as described here.

Class Hierarchy

The main base class is `MapObject`, which models the common behavior of all map objects. Instances of this class should never be directly instantiated, nor the intermediate classes; always create objects of the most precise class (leaf nodes of the class tree).

The hierarchy may be visualized as shown in figure B.1.

Common Behavior for All Types

Each type has a constructor as described above, which sets attributes with the same names as the constructor parameters. Most also have `from_json` constructors which return a new object of the given type based on a dictionay of object attributes such as that unmarshalled from the JSON payload of a server message or read from disk file. Most also have a `to_json` method which does the opposite.

Those with a `record_type` method use that to return the name of the record in the mapper file which would hold the data for that type of object.

All `MapObjects` have a `ObjID` method which returns the ⟨*ID*⟩ attribute of the object. This is provided for compatibility with the API for other languages.

The meanings of the individual attributes are as documented in the server protocol document `gma-mapper-protocol`(7).

Functions

`read_map_file(`⟨*path*⟩`)`
: Reads the file at the named ⟨*path*⟩ as a mapper save file. Returns a list of `MapObjects` and a `MapMetaData` object.

`read_map_meta_data(`⟨*path*⟩`)`
: As `read_map_file`, but only goes as far as reading the metadata, which is returned as the single return value. This stops reading the input file once the meta data has been read.

`load_map_meta_data(`⟨*fileobj*⟩`)`
: As `read_map_meta_data`, but reads from the file-like object ⟨*fileobj*⟩.

`load_map_file(`⟨*fileobj*⟩`, `⟨*meta_only*⟩`=False)`
: As `read_map_file`, but reads from the file-like object ⟨*fileobj*⟩. If ⟨*meta_only*⟩ is true, only the metadata will be read and returned (the object list will be an empty list value).

`write_map_file(`⟨*path*⟩`, `⟨*obj_list*⟩`, `⟨*meta*⟩`)`
: Writes the objects in ⟨*obj_list*⟩ to the file named in ⟨*path*⟩, along with the `MapMetaData` value in ⟨*meta*⟩.

`save_map_file(`⟨*fileobj*⟩`, `⟨*obj_list*⟩`, `⟨*meta*⟩`)`
: As above but writes to the file-like object ⟨*fileobj*⟩.

Map (Graphical) Elements

The `MapElement` subclass describes any kind of graphic element that would be "drawn" on the map canvas. These should never be created directly; instead create an instance of a subclass of `MapElement`.

Creature Tokens

Creatures on the battle map are represented by objects derived from the `CreatureToken` class. These are not `MapElement` objects. Rather than describing a static graphical element of the map, they describe moving creatures who act in that environment.

Never create a `CreatureToken` directly; instead create a `PlayerToken` or `MonsterToken` depending on the creature's role.

Utility Functions

Two functions are provided for translating between map location labels such as those used for creature spawn points and the actual *grid-unit* (x, y) coordinates.

`xy_to_map_label(`$\langle x \rangle$`, `$\langle y \rangle$`)`
: Takes *grid-unit* coordinates $\langle x \rangle$ and $\langle y \rangle$, returning a text label where the x coordinate is represented by an alphabetic value which follows the sequence "`A`", "`B`", ... "`Z`", "`AA`", "`AB`", ... "`AZ`", "`BA`", "`BB`", etc, followed by the numeric y coordinate.

  For example, the grid coordinates (13,42) translate to label "`N42`".

`map_label_to_xy(`$\langle label \rangle$`)`
: This function performs the inverse of the `xy_to_map_label` function. The $\langle label \rangle$ string is interpreted regardless of case.

EXCEPTIONS

This module defines `InvalidMapFile` as a general-purpose exception for errors encountered, and `InvalidMapFileVersion` for the specific case of an unsupported file format version.

ENUMERATED TYPES

The following enumerated types are defined:

`AnchorDirection`
: `CENTER`, `NORTH`, `SOUTH`, `EAST`, `WEST`, `NE`, `NW`, `SW`, `SE`.

`AoEType`
: `CONE`, `RADIUS`, `RAY`.

`ArcModeType`
: `PIESLICE`, `ARC`, `CHORD`.

`ArrowType`
: `NONE`, `FIRST`, `LAST`, `BOTH`.

`CreatyreTypeCode`
: `UNKNOWN`, `MONSTER`, `PLAYER`.

`DashType`
: `SOLID`, `LONG`, `MEDIUM`, `SHORT`, `LONGSHORT`, `LONG2SHORT`.

`FontSlantType`
: `ROMAN`, `ITALIC`.

`FontWeightType`
: `NORMAL`, `BOLD`.

`JoinStyle`
: `BEVEL`, `MITER`, `ROUND`.

`MoveModeType`
: `LAND`, `BURROW`, `CLIMB`, `FLY`, `SWIM`.

SEE ALSO

`gma-socketinterface`(3), `gma-mapper`(5), `gma-mapper`(6), `gma-mapper-protocol`(7).

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

BUGS

Doesn't handle all kinds of values correctly yet, specifically null values and base-64 encoded binary values.

gma-mapper(3)

NAME

gma-mapper – Various functions of interest

SYNOPSIS

```
source gmazones.tcl
ComputedReachMatrix ⟨size_code⟩ ⟨natural⟩ ⟨extended⟩ ⇒ ⟨template⟩
CreatureDisplayedSize ⟨id⟩ ⇒ ⟨size_code⟩
creature_display_zoom ⟨size_code⟩ ⟨dispsize⟩ ⟨zoom⟩ ⇒
⟨effective_zoom⟩
CreatureSizeParams ⟨size_code⟩ ⇒ {⟨category⟩ ⟨natural⟩ ⟨extended⟩
⟨space⟩}
FullCreatureAreaInfo ⟨id⟩ ⇒ {⟨space⟩ ⟨natural⟩ ⟨extended⟩
⟨matrix⟩ ⟨custom⟩}
MatchesStandardTemplate ⟨size_code⟩ ⟨natural⟩ ⟨extended⟩ ⇒
⟨template⟩
MonsterSizeValue ⟨size_code⟩ ⇒ ⟨grids⟩
ReachMatrix ⟨size_code⟩ ⇒ ⟨template⟩
```

DESCRIPTION

This document describes the usage of a number of functions internal to the mapper client, either in the `mapper.tcl` or `gmazones.tcl` source file, but are not in their own package.

Over time, more functions may be added here as it seems useful to have them documented for reference.

`ComputedReachMatrix` ⟨*size_code*⟩ ⟨*natural*⟩ ⟨*extended*⟩
: If the specified values match an existing template, that is returned without going to any more trouble. Otherwise, a template is calculated and returned based on the requested values. The ⟨*natural*⟩ and ⟨*extended*⟩ values are in grid square units.

`CreatureDisplayedSize` ⟨*id*⟩
: Given a creature's ⟨*id*⟩ number, this returns the ⟨*size_code*⟩ that should be used for displaying it. This is based on the creature's `Size` and `DispSize` attributes.

`creature_display_zoom` ⟨*size_code*⟩ ⟨*dispsize*⟩ ⟨*zoom*⟩
: Given a creature's ⟨*size_code*⟩, the size code ⟨*dispsize*⟩ that you want to display them as (temporarily), and the map's current ⟨*zoom*⟩ factor, this returns the zoom factor for the creature token image to use to accomplish this, assuming that creature images are available at zoom

factors 0.25, 0.5, 1, 2, 3, 4, 6, 8, 12, 16, and 32. It will choose the image zoom factor from that list that is not larger than the creature's token size. This means a smaller image may be used.

`CreatureSizeParams` ⟨*size_code*⟩
Given a creature size code in the form ⟨*category*⟩[⟨*nat*⟩][`->`⟨*ext*⟩][`=`⟨*space*⟩], this function parses out the various parameters that may appear in that code. Note that the values for ⟨*nat*⟩, ⟨*ext*⟩, and ⟨*space*⟩ are in units of feet. If the ⟨*size_code*⟩ is invalid, the empty string will be returned. Otherwise the return value is a four-element list consisting of the following elements. Any which were not specified in the ⟨*size_code*⟩ are empty strings in the returned list.

⟨***category***⟩	The size category as a single letter.
⟨***nat***⟩	The natural reach diameter in grid squares.
⟨***ext***⟩	The extended reach diameter in grid squares.
⟨***space***⟩	The creature's occupied space diameter in grid squares.

`FullCreatureAreaInfo` ⟨*id*⟩
This is the main function that will usually be called when managing on-screen creatures. It accepts a creature ⟨*id*⟩ which is used to look up the creature's `CustomReach`, `DispSize`, and `Size`. It then returns the creature's actual distance values as a list with the following elements, or it returns the empty string if it was unable to understand the creature's size.

⟨***space***⟩	The creature's occupied space in grid-square units.
⟨***natural***⟩	The creature's natural reach distance in grid-square units.
⟨***extended***⟩	The creature's extended reach distance in grid-square units.
⟨***matrix***⟩	The creature's threat zone matrix as described for `ReachMatrix`.
⟨***custom***⟩	A dictionary containing the creature's `CustomReach` attribute if there is one; otherwise it is the empty string.

`MatchesStandardTemplate` ⟨*size_code*⟩ ⟨*natural*⟩ ⟨*extended*⟩
Given a ⟨*size_code*⟩ as described above, and the desired ⟨*natural*⟩ and ⟨*reach*⟩ distances in grid squares, this function returns a ⟨*template*⟩ list, as defined in the description of the `ReachMatrix` function, if the values happen to exactly match one of the standard creature space and reach templates. Otherwise, it returns the empty string.

`MonsterSizeValue` ⟨*size_code*⟩
Returns the space occupied by the creature in units of grid squares, which may be a real number. If the ⟨*size_code*⟩ cannot be understood, 0 is returned.

`ReachMatrix` ⟨*size_code*⟩

Given a ⟨*size_code*⟩, this returns a creature area template based on the first character of ⟨*size_code*⟩. If that character does not correspond to a standard size category, the empty string is returned. Otherwise, the template is a 3-element list with the following values:

⟨***natural***⟩ The distance in grid squares of the creature's natural reach zone.

⟨***extended***⟩ The distance in grid squares of the creature's extended reach zone.

⟨***matrix***⟩ a 2D list in row-major order. Each element describes a grid square on the map, with the creature in the center of that matrix. If an element has the value 0, that square is not part of the creature's threat zones. If it is 1, then it is part of the extended area; if 2, it is part of the natural area; if 3, it is part of both.

AUTHOR

Steve Willoughby / steve@madscience.zone.

COPYRGHT

Part of the GMA software suite, copyright © 1992–2025 by Steven L. Willoughby, Aloha, Oregon, USA. All Rights Reserved. Distributed under BSD-3-Clause License.

gma-mapservice(3)

NAME

MapService – GMA battle grid map server (API Description) (DEPRECATED)

SYNOPSIS

```
import MadScienceZone.GMA.Mapper.MapService
PROTOCOL_VERSION

⟨base_name⟩ = strip_creature_base_name(⟨name⟩)

⟨dp⟩ = DicePreset(⟨name⟩, ⟨description⟩, ⟨roll_list⟩)
⟨dp⟩.description
⟨dp⟩.name
⟨dp⟩.roll_list

⟨dc⟩ = DicePresetCollection(⟨filename⟩)
⟨dc⟩.store(⟨user⟩, ⟨presetlist⟩)
⟨dc⟩.fetch(⟨user⟩)
⟨dc⟩.load_presets()
⟨dc⟩.save_presets()

⟨event⟩ = MapEvent(⟨rawline⟩, ⟨sequence⟩, ⟨obj_class⟩=None,
⟨obj_id⟩=None)
⟨event⟩.event_type
⟨event⟩.fields
⟨event⟩.key
⟨event⟩.obj_class
⟨event⟩.obj_id
⟨event⟩.raw
⟨event⟩.sequence
⟨event⟩.guess_class(⟨attrs⟩)
```

```
⟨disp⟩ = MapIoDispatcher(⟨host⟩=, ⟨port⟩=2323, ⟨logfile⟩=None,
⟨secret⟩=None, ⟨gm_secret⟩=None, ⟨dice_preset⟩=None)
⟨disp⟩.map_list
⟨disp⟩.state
⟨disp⟩.master_mode
⟨disp⟩.dice_preset_obj
⟨disp⟩.accept_incoming(⟨flag⟩)
⟨disp⟩.dump_state()
⟨disp⟩.save_state(⟨filename⟩, ⟨force⟩=False)
⟨disp⟩.load_state(⟨filename⟩)
⟨disp⟩.sync_on_connect(⟨flag⟩)
⟨disp⟩.send_to_new(⟨message_list⟩)
⟨disp⟩.handle_accept()
⟨disp⟩.remove(⟨connection⟩)
⟨disp⟩.check_if_master_connected()
⟨disp⟩.ping_all()
⟨disp⟩.send_to_all(⟨message⟩)
⟨n⟩ = ⟨disp⟩.connection_count()
⟨disp⟩.stop()
⟨events⟩ = EventCollection()
⟨events⟩.events
⟨events⟩.image_id
⟨events⟩.save_needed
⟨events⟩.reset()
⟨events⟩.load_identities(⟨filename⟩)
⟨events⟩.save(⟨save_file⟩)
⟨iterator⟩ = ⟨events⟩.sync()
⟨events⟩.dump_state()
⟨to_requester⟩, ⟨to_peers⟩ = ⟨events⟩.accept(⟨raw⟩, ⟨peers⟩=None,
⟨me⟩=None, ⟨presets⟩=None)

⟨client⟩ = MapIoClientConnection(⟨socket_obj⟩, ⟨addr⟩,
⟨dispatcher⟩, ⟨authenticator⟩)
⟨client⟩.authenticated
⟨client⟩.authenticator
⟨client⟩.client_addr
⟨client⟩.dispatcher
⟨client⟩.user
⟨client⟩.write_only
⟨client⟩.push(⟨message⟩)
⟨client⟩.found_terminator()
⟨client⟩.collect_incoming_data(⟨data⟩)
⟨client⟩.handle_close()
```

```
⟨service⟩ = MapService(⟨host⟩=, ⟨port⟩=2323, ⟨secret⟩=None,
⟨logfile⟩=None, ⟨gm_secret⟩=None, ⟨dice_preset⟩=None)
⟨service⟩.daemon_running
⟨service⟩.dice_preset
⟨service⟩.dispatcher
⟨service⟩.host
⟨service⟩.port
⟨service⟩.secret
⟨service⟩.gm_secret
⟨service⟩.logfile
⟨service⟩.comm_thread
⟨service⟩.save_state(⟨filename⟩, ⟨force⟩=False)
⟨service⟩.load_state(⟨filename⟩)
⟨bool⟩ = ⟨service⟩.is_alive()
⟨service⟩.sync_on_connect(⟨flag⟩)
⟨n⟩ = ⟨service⟩.connection_count()
⟨service⟩.shutdown()
⟨service⟩.stop()
⟨service⟩.worker_thread()
⟨service⟩.ping_all()
⟨service⟩.send_to_all(⟨message⟩)
⟨service⟩.send_to_new(⟨message_list⟩)
```

DESCRIPTION

This module is deprecated. The current implementation of the map service is the `gma-go-server`(6) program from the Go-GMA package.

This module implements the Mapper service used by `gma-mapper`(6) clients to keep in sync with each other and to facilitate other communication between them and the other GMA components.

If invoked as a stand-alone Python script, it behaves as documented in `gma-mapservice`(6), starting an instance of a running server according to the options passed on the command line.

It is also usable as a module inside another program (theoretically) that would want to run a server as part of its operation. Documenting the API here serves to provide implementation notes even if another program doesn't actually use it to run an embedded server.

GLOBALS

The global variable `PROTOCOL_VERSION` indicates the version of the mapper protocol implemented by this module.

There is also a `strip_creature_base_name()` function used internally (but could be called from outside) that takes a creature name as tracked by the mapper, returning the displayed part of that name. That is, it strips

away everything up to and including the first equals sign it finds and returns the result.

SERVICE OBJECT

The `MapService` object orchestrates the server operation. A typical server would consist of creating one of these objects, which we'll call ⟨*service*⟩ here, waiting for the dispatch daemon to start, calling ⟨*service*⟩`.load_state()` to load pre-existing state back into the server, and then reading the initialization file, performing the following functions:

1. If syncing of clients is desired every time they connect (the reference implementation does this upon finding a `SYNC` command in the initialization file), arrange this by calling ⟨*service*⟩`.sync_on_connect(True)`
2. If loading of image identities is desired (as when the `LOAD` command is in the initialization file), do so by calling

   ⟨*service*⟩`.dispatcher.state.load_identities()`

   with the name of the file to load
3. For any other command we want to send all clients upon connection, add the command string to the greeting by passing it to

   ⟨*service*⟩`.send_to_new()`
4. In addition to (3), if the command is `AC`, to add a character to the mapper menu, store class `'P'` in

   ⟨*service*⟩`.dispatcher.state.class_by_id[`⟨*obj_id*⟩`]`

   and store the character's object `ID` ⟨*obj_id*⟩ in

   ⟨*service*⟩`.dispatcher.state.id_by_name[`⟨*name*⟩`]`

   (Yes, that really should be coded into a method to make that easier to do.)

Next, the server enables incoming connections by calling

⟨*service*⟩`.accept_incoming(True)`

and enters an event loop wherein it periodically sends a keepalive signal to all clients via ⟨*service*⟩`.ping_all()` and calls ⟨*service*⟩`.save_state()` as necessary.

Meanwhile, the I/O daemon thread started internally in ⟨*service*⟩ will interact with the client connections.

Constructor

A new ⟨*service*⟩ object is constructed as:

⟨*service*⟩=`MapService(`⟨*host*⟩=`''`, ⟨*port*⟩=`2323`, ⟨*secret*⟩=`None`, ⟨*logfile*⟩=`None`, ⟨*gm_secret*⟩=`None`, ⟨*dice_preset*⟩=`None)`

This will launch the I/O service daemon as a new thread. The ⟨*host*⟩ and ⟨*port*⟩ parameters specify the TCP port to listen on for incoming connections. (The ⟨*host*⟩ value may be left blank.) The ⟨*secret*⟩ and ⟨*gm_secret*⟩ give the passwords used for authentication, if authentication is needed for this server. If ⟨*logfile*⟩ is provided, it names a file to which logging information will be appended. If dice presets are being used, the object managing them is passed as ⟨*dice_preset*⟩.

Once the daemon thread is running, it will create a `MapIoDispatcher` object which is accessible as ⟨*service*⟩`.dispatcher` (but take care to avoid resource conflicts between threads; for the most part, the daemon should be doing the work with that object).

Methods

⟨*service*⟩`.save_state(`⟨*filename*⟩`)`
: This saves the current game state to the specified file by calling ⟨*service*⟩`.dispatcher.save_sta`

⟨*service*⟩`.load_state(`⟨*filename*⟩`)`
: This loads the game state from the specified file by calling ⟨*service*⟩`.dispatcher.load_state(⟨f`

⟨*bool*⟩=⟨*service*⟩`.is_alive()`
: Returns true if the daemon thread is still running. This does not necessarily mean it's ready to interact with clients. To see that, look at the value of ⟨*service*⟩`.daemon_running` instead.

⟨*service*⟩`.sync_on_connect(`⟨*flag*⟩`)`
: If ⟨*flag*⟩ is true, the server will sync the game state with every connecting client. Otherwise, it will only do so on request from the client. This calls ⟨*service*⟩`.dispatcher.sync_on_connect(`⟨*flag*⟩`)`.

⟨*n*⟩=⟨*service*⟩`.connection_count()`
: Returns the number of connected clients.

⟨*service*⟩`.shutdown()`
: Notifies the clients (via comment) that the server is shutting down and then shuts down as per the ⟨*service*⟩`.stop()` method.

⟨*service*⟩`.stop()`
: Shuts down the client connections and dispatcher daemon.

⟨*service*⟩`.worker_thread()`
: This is the daemon process. It is intended to be started internally in a new thread rather than called directly.

⟨*service*⟩`.ping_all()`
: Ping all connected clients to see that they are still alive. Calls ⟨*service*⟩`.dispatcher.ping_all()`.

⟨*service*⟩`.send_to_all(`⟨*message*⟩`)`
: Send ⟨*message*⟩ to all connected clients. Calls ⟨*service*⟩`.dispatcher.send_to_all(`⟨*message*⟩`)`.

⟨*service*⟩`.send_to_new(`⟨*message_list*⟩`)`
: Add the list of commands in ⟨*message_list*⟩ to the initial set of commands sent to newly connecting clients. Calls ⟨*service*⟩`.dispatcher.send_to_new(`⟨*message_list*⟩`)`.

Attributes

⟨*service*⟩`.host`
: The host interface we're listening on.

⟨*service*⟩`.port`
: The host port we're listening on.

⟨*service*⟩`.dispatcher`
: The `MapIoDispatcher` object we're using to handing client communications.

⟨*service*⟩`.daemon_running`
: A boolean; true if the I/O daemon thread is running and is running its I/O handling loop.

⟨*service*⟩`.dice_preset`
: The object managing die-roll presets.

I/O OPERATIONS

When the ⟨*service*⟩ object is up and running, its `worker_thread()` enters an `asyncore.loop()` event loop. This will call back to the ⟨*service*⟩`.dispatcher` object to handle received messages from clients. This is a `MapIoDispatcher` object (which is a subclass of `asyncore.dispatcher`).

⟨*disp*⟩`=MapIoDispatcher(`⟨*host*⟩`='', `⟨*port*⟩`=2323, `⟨*logfile*⟩`=None,`
⟨*secret*⟩`=None, `⟨*gm_secret*⟩`=None, `⟨*dice_preset*⟩`=None)`

Construct a new dispatcher object. This is generally done by the `MapService` object which creates a dispatcher for its own use internally. The parameters are as described for that class.

Methods

⟨*disp*⟩`.accept_incoming(`⟨*flag*⟩`)`
: If ⟨*flag*⟩ is `True`, the dispatcher will accept incoming connections from clients. Otherwise, any connection from a client will be turned away with a message that the server isn't ready.

⟨*disp*⟩`.dump_state()`
: Print to standard output a description of the internal state of the game and current connections to the server.

⟨*disp*⟩`.save_state(`⟨*filename*⟩`,` ⟨*force*⟩`=False)`
: Saves the game state to the specified file. The previous file with that name is backed up to a file with the suffix `.old`. If the game state doesn't need to be saved (as determined by the ⟨*disp*⟩`.state.save_needed` attribute), nothing will be done by this method, unless the ⟨*force*⟩ parameter is `True`.

⟨*disp*⟩`.load_state(`⟨*filename*⟩`)`
: Load the saved state from the specified file. This will become the current game state, replacing all state data currently in effect.

⟨*disp*⟩`.sync_on_connect(`⟨*flag*⟩`)`
: If ⟨*flag*⟩ is true, then any new clients which connect will automatically see the result of a `SYNC` command. Otherwise, this will not be done.

⟨*disp*⟩`.send_to_new(`⟨*message_list*⟩`)`
: Adds each message string in ⟨*message_list*⟩ to the set of initial messages sent to all connecting clients.

⟨*disp*⟩`.add_to_new(`⟨*message*⟩`)`
: Add the single ⟨*message*⟩ string to the set of initial messages sent to all connecting clients.

⟨*disp*⟩`.handle_accept()`
: The main I/O loop will invoke this method to handle a new client connection. It completes the socket connection, sends initial greetings, and initiates the authentication negotiation if configured with passwords. If the server is in master mode (i.e., any client connected has identified itself as a master client—normally this is only done by the `GM` console), the server will instruct the client to disable its toolbar.

⟨*disp*⟩`.remove(`⟨*connection*⟩`)`
: Given a ⟨*connection*⟩ (a `MapIoClientConnection` object), this connection is terminated and removed from the list of clients the server is interacting with. If this was the last master client, the server releases master mode from the clients.

⟨*disp*⟩`.check_if_master_connected()`
: Looks at the current set of connected clients. The ⟨*disp*⟩`.master_mode` attribute will be set to reflect whether any of them are master clients or not.

⟨*disp*⟩`.ping_all()`
: Send a `MARCO` command to all connected clients. If a client has not authenticated by the time `ping_all()` pings it three times, that client

is disconnected from the server. This helps remove stray connections which aren't actual GMA clients or don't do anything after connecting.

⟨*disp*⟩`.send_to_all(`⟨*message*⟩`)`
Send ⟨*message*⟩ to all listening clients. Calls ⟨*client*⟩`.push(`⟨*message*⟩`)` for each such client.

⟨*n*⟩=⟨*disp*⟩`.connection_count()`
Returns the number of connected clients.

⟨*disp*⟩`.stop()`
Remove all clients and close down the dispatcher.

Attributes

⟨*disp*⟩`.map_list`
A list of connected clients, each as a `MapIoClientConnection` object.

⟨*disp*⟩`.state`
The game state. This is a `EventCollection` object.

⟨*disp*⟩`.master_mode`
This is true if one of the clients is a master tool.

⟨*disp*⟩`.dice_preset_obj`
The object managing die roll presets for users.

INDIVIDUAL CLIENTS

Each client connection is represented by a `MapIoClientConnection` object, which is a subclass of `asynchat.async_chat`. These are created automatically by the dispatcher object as new clients connect to the server.

⟨*client*⟩=`MapIoClientConnection(`⟨*socket_obj*⟩, ⟨*addr*⟩, ⟨*dispatcher*⟩, ⟨*authenticator)*⟩

Creates a new client connection object. The parameters are, respectively, the socket object for the underlying TCP/IP socket, the client's IP address and port, the dispatcher object which is managing the connections, and an `Authenticator` object for this client session.

Methods

⟨*client*⟩`.push(`⟨*message*⟩`)`
Send ⟨*message*⟩ to the connected client. This method encodes the string argument as a UTF-8 byte sequence and ensures that it is a valid Tcl list (but does not further verify the correctness of the message in terms of the protocol). A newline is appended to the message string as it is sent.

⟨*client*⟩`.found_terminator()`
: This method is called (due to the semantics of `asyncore` communications) automatically when a complete line of data has been seen coming in from the client. The request from the client is carried out directly in this method or by calling `dispatcher.state.accept()` to process it at that level.

⟨*client*⟩`.collect_incoming_data(`⟨*data*⟩`)`
: Automatically called from `asyncore` to receive incoming data from the client socket.

⟨*client*⟩`.handle_close()`
: Called automatically to handle the event of a client connection closing.

Attributes

⟨*client*⟩`.authenticated`
: True if the client has successfully authenticated to the server.

⟨*client*⟩`.authenticator`
: The `Authenticator` object used for negotiating with the client.

⟨*client*⟩`.client_addr`
: The IP address and port for this client.

⟨*client*⟩`.dispatcher`
: The `MapIoDispatcher` object which is managing all the clients.

⟨*client*⟩`.user`
: The logged-in username for this client.

⟨*client*⟩`.write_only`
: This client intends to only send commands to us and is not listening for any responses. The server won't send any messages to it.

EVENTS

An event is an incoming command received from a client. Each individual event is represented in an instance of a `MapEvent` object.

⟨*event*⟩`=MapEvent(`⟨*rawline*⟩`,` ⟨*sequence*⟩`,` ⟨*obj_class*⟩`=None,`
⟨*obj_id*⟩`=None)`

Constructor for a `MapEvent` object. The parameters are:

⟨*rawline*⟩
: The raw input line received from the client. This will be a single event string with no newline terminator or, in the case of multi-line commands such as `LS`, a string holding all lines received for a single such event, with newlines between the individual lines.

⟨*sequence*⟩
: The sequence number for this event, assigned by the caller, to distinguish this event from others.

⟨*obj_class*⟩
: If specified, gives the class of object this represents, which may be one of the following:

  `E`
  : Map element such as a polygon or line.

  `M`
  : Monster.

  `P`
  : Player.

  If no ⟨*obj_class*⟩ is given, the constructor will attempt to make its best guess based on the contents of the event.

⟨*obj_id*⟩
: The object's unique ID as known to the mapper clients.

  This function generates the attribute ⟨*event*⟩`.key` which provides an identifier of the event for the purposes of tracking state. The server will remember the most recent event with any given key, so that if a client requests a copy of the current game state, they won't get a replay of everything that's happened since the server started; they will only receive the current state of the things the server saw get changed. If this attribute is set to `None`, that means this is not an event that needs to be tracked at all.

Methods

⟨*event*⟩`.guess_class(`⟨*attrs*⟩`)`
: Given a set of attributes in ⟨*attrs*⟩, guess the class of the object whose attributes are being changed by the event. This sets the ⟨*event*⟩`.obj_class` attribute if it's able to do so.

⟨*event*⟩`.event_type`
: The event type is the command part of the event, such as `AC`, `MARCO`, or `OA`.

⟨*event*⟩`.fields`
: The list of parameters to the event as a list of values. Element 0 of this list is the event type as found in ⟨*event*⟩`.event_type`.

⟨*event*⟩`.key`
: The key used to determine whether (and for how long) this event will be considered part of the persistent game state, as described above.

⟨*event*⟩`.obj_class`
: The object class as already described.

⟨*event*⟩`.obj_id`
: The object ID as already described.

⟨*event*⟩`.raw`
: The raw event text as received.

⟨*event*⟩`.sequence`
: The assigned event sequence number.

The game state is represented by a collection of these event objects, managed by an instance of `EventCollection` as described below.

⟨*events*⟩`=EventCollection()`
: Create a new game state object with no tracked content.

⟨*events*⟩`.load_identities(`⟨*filename*⟩`)`
: Load image identity information from the given file. The file must be in `gma-mapper`(5) format. This only looks for "I" lines from that file, and only those which give server-side image definitions. This is automatically invoked during server start-up if the server's initial commands includes a `LOAD` command.

⟨*events*⟩`.save(`⟨*save_file*⟩`)`
: Save the game state to the named file.

⟨*events*⟩`.add_event(`⟨*raw*⟩`,` ⟨*obj_class*⟩`=None,` ⟨*obj_id*⟩`=None)`
: Adds the raw received data to our game state. This creates a new `MapEvent` object and stores it in the ⟨*events*⟩`.events` dictionary if it is an event we care about tracking.

⟨*events*⟩`.sync()`
: This yields an iterator which will iterate over the set of events in sequential order.

⟨*events*⟩`.dump_state()`
: Prints to standard output a description of the game state.

⟨*to_requester*⟩`,` ⟨*to_peers*⟩`=`⟨*events*⟩`.accept(`⟨*raw*⟩`,` ⟨*peers*⟩`=None,` ⟨*me*⟩`=None,` ⟨*presets*⟩`=None)`
: This is the method that accepts an incoming event from a client. The command in ⟨*raw*⟩ is interpreted and acted upon. New events are added to the game state by calling ⟨*events*⟩`.add_event()` as needed. Where special handling is required, it will communicate directly with the list of all connected peer connections (in ⟨*peers*⟩) of which the connection which issued this command is passed as ⟨*me*⟩. The object managing the collection of die roll presets is passed in as the ⟨*presets*⟩

parameter. The text to be sent back to the requesting connection and to all other connections, respectively, is returned. If either or both of these strings are `None`, then no message needs to be sent to those connections.

⟨*events*⟩`.image_id`
: A dictionary mapping an image ID formatted as "⟨*imageID*⟩^A⟨*zoom*⟩^A⟨*filename*⟩" to the server-side filename (sans "@") for that file. This is used to answer client queries for where to find images. If the client is looking for one in this dictionary, the server will privately give that answer to it. Otherwise, it will ask the other clients to see if they can answer the query.

⟨*events*⟩`.events`
: A dictionary mapping event keys to their `MapEvent` objects. Since the event key is used as the dictionary key, each one received will overwrite the previous event with the same key, thus retainig only the most recent event of a given type.

⟨*events*⟩`.save_needed`
: A boolean value, true if the game state needs to be saved to disk.

DIE ROLLING SUPPORT

The `DicePresetCollection` object holds all of the presets for users of the server. It manages a collection of `DicePreset` objects, which represent each individual preset.

⟨*dp*⟩`=DicePreset(`⟨*name*⟩`,` ⟨*description*⟩`,` ⟨*roll_list*⟩`)`
: Create a `DicePreset` structure. The parameters may be accessed as attributes with the same names.

⟨*dc*⟩`=DicePresetCollection(`⟨*filename*⟩`)`
: Create a collection of dice presets, persistently stored in the named file. As part of the construction of the object, this file will be opened and its contents loaded as by the ⟨*dc*⟩`.load_presets()` method.

⟨*dc*⟩`.store(`⟨*user*⟩`,` ⟨*presetlist*⟩`)`
: Store the list of `DicePreset` objects in ⟨*presetlist*⟩ for the named ⟨*user*⟩. This replaces the set that user had previously stored, if any.

⟨*presetlist*⟩`=`⟨*dc*⟩`.fetch(`⟨*user*⟩`)`
: Retrieves a list of `DicePreset` objects which are in the storage for the named ⟨*user*⟩, or `None` if that user does not have any.

⟨*dc*⟩`.load_presets()`
: Load the stored presets from disk file.

⟨*dc*⟩`.save_presets()`
: Save the presets to disk file. This makes a backup copy of the file with a `.old` suffix first.

SEE ALSO

`gma-authenticator`(3), `gma`(6), `gma-mapper`(5), `gma-mapper`(6), `gma-server`(6), `gma-go-server`(6).

AUTHOR

Steve Willoughby / steve@madscience.zone.

BUGS

The code which drops unauthenticated connections doesn't correctly consider if the server is using authentication at all, so until that is fixed, it is effectively required for the server to use passwords.

gma-markupline(3)

See gma-markuptext(3), p. 298.

gma-markuptext(3)

NAME

MarkupText – Rendering of text with simple markup language

SYNOPSIS

```
import MadScienceZone.Common.MarkupText

ps_str_raw(⟨v⟩) ⇒ ⟨str⟩
ps_str(⟨v⟩) ⇒ ⟨str⟩

link_type(⟨target⟩) ⇒ ⟨str⟩
link_target(⟨target⟩) ⇒ ⟨str⟩

fragments_text(⟨fragments⟩) ⇒ ⟨str⟩
fragments_lines(⟨fragments⟩) ⇒ [⟨str⟩, ⟨...⟩]

markupline(⟨source⟩, ⟨onestyle⟩=False, ⟨allow_line_breaks⟩=True)
⇒ ⟨fragments⟩
consolidate_fragments(⟨fragments⟩) ⇒ ⟨fragments⟩

html2markup(⟨html⟩, ⟨links⟩=None, ⟨allow_unicode⟩=True) ⇒ ⟨str⟩

⟨m⟩ = MarkupText(⟨source⟩=None)
⟨m⟩.source
⟨m⟩.parse(⟨source⟩)
⟨text⟩ = ⟨m⟩.render(⟨formatter⟩=None)

⟨mf⟩ = MarkupFragment(⟨text⟩, ⟨is_bold⟩=False, ⟨is_italic⟩=False)
⟨mf⟩.is_bold
⟨mf⟩.is_italic
⟨mf⟩.text

⟨mf⟩ = MarkupFragmentLineBreak()

⟨mf⟩ = MarkupFragmentLink(⟨link_target⟩, ⟨link_text⟩)
⟨mf⟩.link_target
⟨mf⟩.link_text
```

```
⟨f⟩ = MarkupTextFormatter()
⟨f⟩.new_par()
⟨f⟩.process(⟨text⟩)
⟨f⟩.toggle_ital()
⟨f⟩.set_bold(⟨on⟩=True)
⟨f⟩.set_ital(⟨on⟩=True)
⟨f⟩.new_line()
⟨f⟩.table(⟨tableobj⟩)
⟨f⟩.reference(⟨display_name⟩, ⟨link_name⟩)
⟨f⟩.bullet_list_item(⟨level⟩)
⟨f⟩.enum_list_item(⟨level⟩, ⟨counter⟩)
PsFormFormatter(⟨ps_escape⟩=None, ⟨compact⟩=False)

HTMLFormatter()

NullFormatter()
```

DESCRIPTION

A `MarkupText` object represents a string value with formatting. The markup source syntax is intentionally simple and inspired by (but not directly based on or derived from) Creole. Once loaded from markup source, the original source may be retrieved, or the plain text of the string (sans formatting) can be obtained. Various formatted output translators may be created to render the final text.

⟨*m*⟩ = `MarkupText(`⟨*source*⟩`)`

The ⟨*source*⟩ is optional. It may be provided later by calling the `parse()` method. (If provided to the constructor, the constructor simply calls that method for you up front.)

Once constructed, the original source string is available in the ⟨*m*⟩`.source` attribute.

The text being marked up may include a number of Unicode characters and ASCII symbols which represent special characters. Output formats which are Unicode-compatible may be able to display all input characters, but only the codes in Table B.1 are guaranteed to work with the PostScript renderer.

Notes:

1 At the end of a number, use an underscore before the fraction. For example, "`12_1/2`" would appear as "12$\frac{1}{2}$".

2 To get a stand-alone multiplication sign, enclose it in square brackets as shown above. If a letter `x` appears adjacent to a number, as in "`12x`" or "`x12`", then the multiplication sign is used instead of the `x`.

Unicode Character	Markup	Output Symbol	Notes
U+00A7	[S]	§	
U+00A9	[c]	©	
U+00AB	[<<]	«	
U+00AE	[R]	®	
U+00B0	^o	°	
U+00B1	+/-	±	
U+00B7/U+2022	^.	•	
	[0]	^{0}	
U+00B9	[1]	^{1}	
U+00B2	[2]	^{2}	
U+00B3	[3]	^{3}	
	[4]	^{4}	
	[5]	^{5}	
	[6]	^{6}	
	[7]	^{7}	
	[8]	^{8}	
	[9]	^{9}	
U+00BB	[>>]	»	
U+00BC	1/4	¼	1
U+00BD	1/2	½	1
U+00BE	3/4	¾	1
U+00C4		Ä	
U+00C6	AE	Æ	
U+00D7	[x]	×	2
U+00E4		ä	
U+00E6	ae	æ	
U+00F7	[/]	÷	
U+2012	-	− minus sign	3
U+2013	--	– en dash	
U+2014	---	— em dash	
U+2018	‘	‘	
U+2019	’	’	
U+201C	‘‘	“	
U+201D	’’	”	
U+2020		†	
U+2021		‡	
U+FB00		ff	
U+FB01		fi	
U+FB02		fl	
U+FB03		ffi	
U+FB04		ffl	

Table B.1: Unicode and Markup for Special Characters

3 In markup, a hyphen immediately before a digit is converted to a minus sign.

Functions

`html2markup(`⟨*html*⟩`,` ⟨*links*⟩`=None,` ⟨*allow_unicode*⟩`=True)`

Given a string ⟨*html*⟩ encoded with HTML, output an equivalent string encoded with our markup language. If ⟨*allow_unicode*⟩ is `True`, leave Unicode characters in the output on the assumption that the consumer of the markup string can interpret them correctly. Otherwise, they are converted to markup special character codes where possible, leaving any Unicode characters intact that do not have an equivalent markup code.

In order to provide the flexibility to allow the source data to be supplied already in markup format on an individual basis, `html2markup` looks for the string "`~~gma~~`" at the very beginning of the ⟨*html*⟩ string. If present, the remainder of the ⟨*html*⟩ string is simply returned on the assumption that it's already in markup format. For completeness, if the ⟨*html*⟩ string begins with "`~~html~~`", that prefix is removed before processing. Any other prefix in the form "`~~`⟨*word*⟩`~~`" raises an error.

If ⟨*links*⟩ is present and not `None`, it provides a list of known strings that should be hyperlinks in the output. Each element of ⟨*links*⟩ is either a string or a tuple of strings. If a single string, it names the text to be made into a hyperlink as `[[`⟨*string*⟩`]]` in the output. If a 2-tuple (⟨*name*⟩, ⟨*link*⟩), the first element is the string to be replaced with the link `[[`⟨*link*⟩`|`⟨*name*⟩`]]`. If a 3-tuple (⟨*name*⟩, ⟨*anything*⟩, ⟨*url*⟩), then the link is to an external URL (not yet implemented).

`ps_str_raw(`⟨*v*⟩`)`

Escapes the value passed in as ⟨*v*⟩ so that it may be printed as a PostScript string (i.e., between parentheses in PostScript source code). If ⟨*v*⟩ is a `float` or `int` type value, it is converted to a string with commas inserted every three digits.

`ps_str(`⟨*v*⟩`)`

As `ps_str_raw()`, but also transforms the Unicode characters and markup sequences listed above to PostScript codes that will print the intended characters. This assumes the use of the GMA PostScript preamble to set up the correct font encodings.

`markupline(`⟨*source*⟩`,` ⟨*onestyle*⟩`=False,` ⟨*allow_line_breaks*⟩`=True)`

Accepts a ⟨*source*⟩ string in markup format, returning information about it useful to output rendering routines. This is intended to handle markup in restricted areas such as table fields and section headers where not all formatting controls may be available, and where the area is self-contained.

The ⟨*source*⟩ string may include \\ to indicate line breaks (ignored if ⟨*allow_line_breaks*⟩ is `False`), ** and // to indicate font changes, and `[[...]]` to indicate hyperlinks.

The return value is a list of `MarkupFragment`, `MarkupFragmentLineBreak`, and `MarkupFragmentLink` objects. If ⟨*onestyle*⟩ is `True`, the parser will assume that the output medium can only handle a single style for the entire area. In this case, it will determine what is likely the best single format to use. If a link exists in the source, the entire output will be that link (including the full text as the link text). Multiple links are not allowed, so if more than one exists, none will appear in the output as links.

`consolidate_fragments(`⟨*fragments*⟩`)`
: Consolidates a list of fragments to a new list just as described for `markupline` with the ⟨*onestyle*⟩ parameter `True`.

`link_target(`⟨*target*⟩`)`
: Given a link target which may be in the form ⟨*type*⟩:⟨*targ*⟩ this returns the ⟨*targ*⟩ portion of the string. If no colon is found in ⟨*target*⟩, the entire string is returned.

`link_type(`⟨*target*⟩`)`
: Given a link target which may be in the form ⟨*type*⟩:⟨*targ*⟩ this returns the ⟨*type*⟩ portion of the string, or `None` if no type is explicitly specified.

`fragments_text(`⟨*fragments*⟩`)`
: Given a list of fragments as returned by `markupline()`, this function returns a single string consisting of the text content of those fragments, without formatting or line breaks.

`fragments_lines(`⟨*fragments*⟩`)`
: This function works as `fragments_text()` but rather than merging multiple lines into a single string, it returns them as a list of individual strings.

Methods

⟨*m*⟩`.parse(`⟨*source*⟩`)`
: Accept an input text string in ⟨*source*⟩ which may contain our markup syntax, doing some initial parsing as needed in preparation for rendering.

⟨*text*⟩=⟨*m*⟩`.render(`⟨*formatter*⟩`)`
: Render the parsed input source according to the ⟨*formatter*⟩ provided. Different formatters may be provided to format the text suitably to different media as described below. The resulting text is returned. If no formatter is specified, a plain text rendering is performed.

FORMATTERS

The object responsible for taking the digested markup text and emitting a representation of it in some particular format such as HTML or PostScript is embodied in a `MarkupTextFormatter` object. This is the virtual base class, however. Rather than using one of these directly, you should use a subclass of `MarkupTextFormatter` such as one of the following.

These objects provide the following methods:

`new_par()`
: Insert a paragraph break.

`process(`⟨*text*⟩`)`
: Format the ⟨*text*⟩ with the currently-in-effect formatting settings.

`finalize()`
: End the rendering of input text, returning the formatted output.

`toggle_ital()`
: If Italics are currently on, turn them off, and vice versa.

`set_bold(`⟨*on*⟩`=True)`
: If ⟨*on*⟩ is true, enable boldface; otherwise disable it.

`set_ital(`⟨*on*⟩`=True)`
: If ⟨*on*⟩ is true, enable Italics; otherwise disable it.

`new_line()`
: Insert a line break.

`table(`⟨*tableobj*⟩`)`
: Insert a table as described by the table object.

`reference(`⟨*display_name*⟩`,` ⟨*link_name*⟩`)`
: Insert a hyperlink.

`bullet_list_item(`⟨*level*⟩`)`
: Insert a list item.

`enum_list_item(`⟨*level*⟩`,` ⟨*counter*⟩`)`
: Insert a numbered list item. The ⟨*counter*⟩ value must be incremented by the caller; this only formats the item.

PostScript

The `PsFormFormatter` class renders text in PostScript with the assumption that the GMA PostScript preamble has already been sent to the output. The constructor is: `PsFormFormatter(`⟨*ps_escape*⟩`=None,` ⟨*compact*⟩`=False)`

If ⟨*ps_escape*⟩ is provided, it is a function which will take a string value and return that value appropriately escaped for PostScript. It does not need

to add the parentheses around the value, just make sure what's inside will be legal. The default action will be to simply put a backslash in front of any \, (, and) characters found.

If ⟨*compact*⟩ is true, paragraphs will be smushed together to conserve space.

HTML

The `HTMLFormatter` class renders its output in `HTML` suitable for display in web browsers. The constructor takes no parameters.

NULL

There is also a `NullFormatter` class which can be used to render the text as regular plain text. Its constructor takes no parameters.

MARKUP SYNTAX

The markup format is described in `gma-markup`(5).

SEE ALSO

`gma-markup`(5), `gma-markup`(6), `form-preamble`(7), `gma-preamble`(7).

AUTHOR

Steve Willoughby / steve@madscience.zone.

BUGS

Currently, `html2markup` only substitutes hyperlinks for text that is between `<i>` and `</i>` tags, which is fine for named spells but not for things like feats and skills.

The idea of linking to external `URL`s is not really implemented yet, so the reference above to that kind of link is conceptual and subject to change.

Support needs to be added to allow minimal formatting to appear inside table cells.

It would be good to make a way to have tables reflow text within the cells.

gma-markuptextformatter(3)

See gma-markuptext(3), p. 298.

gma-metricize_length(3)

See gma-weightsandmeasures(3), p. 351.

gma-metricize_weight(3)

See gma-weightsandmeasures(3), p. 351.

gma-name_generator_factory(3)

See gma-pfnamegenerator(3), p. 306.

gma-nullformatter(3)

See gma-markuptext(3), p. 298.

gma-ounces_to_grams(3)

See gma-weightsandmeasures(3), p. 351.

gma-pfnamegenerator(3)

NAME

PFNameGenerator – Random Pathfinder Name Generation

SYNOPSIS

```
import MadScienceZone.GMA.PFNameGenerator
⟨list⟩ = all_culture_names()

⟨obj⟩ = name_generator_factory(⟨culture⟩, ⟨constructor_args...⟩)

⟨obj⟩ = ⟨culture_class⟩(⟨gender⟩=None, ⟨start⟩=None, ⟨num⟩=None,
⟨lmin⟩=5, ⟨lmax⟩=10, ⟨generator⟩=None)
⟨list⟩ = ⟨obj⟩.genders_supported()
⟨bool⟩ = ⟨obj⟩.has_surnames()
⟨int⟩ = ⟨obj⟩.default_min(⟨gender⟩)
⟨int⟩ = ⟨obj⟩.default_max(⟨gender⟩)
⟨list⟩ = ⟨obj⟩.generate(⟨gender⟩=None, ⟨start⟩=None, ⟨num⟩=None,
⟨lmin⟩=None, ⟨lmax⟩=None, ⟨surnames⟩=True)
```

DESCRIPTION

This module implements random name generation. It builds names according to letter and phrase patterns representative of naming conventions used by various Golarion cultures. Each culture is modeled in a subclass of `PFNameGenerator.Culture` (e.g., `PFNameGenerator.Culture.Azlanti` for the Azlanti people). While you could construct a culture object by invoking its specific class constructor, the usual method is to call

`name_generator_factory()`,

passing it the name of the culture and any parameters you wish to pass to the constructor. The list of culture names supported by the factory function are available by calling the module function `all_culture_names()`.

Any of the parameters accepted by the `generate()` method may be passed to the constructor. If so, they are used as defaults for calls to `generate()` where those parameters are not specified at that time.

The culture object (here called ⟨*obj*⟩) has the following methods:

`genders_supported()`

> Returns a list of genders available for this culture. Genders are just the set of varieties of names that a culture generates for different people. They may follow a person's gender, or depending on culture, may be influenced by other things such as social status, etc. For our purposes they are merely ways to specify common varieties of names. The

values returned define what is allowed to be passed as the ⟨*gender*⟩ parameter to other methods. Typically, the set of values includes

`F`
: female names

`M`
: male names

`S`
: surnames

`has_surnames()`
: Returns a boolean value indicating if the culture generates surnames. If `False`, then only given names are available.

`default_min(`⟨*gender*⟩`)`
: For the given ⟨*gender*⟩, returns the culture's default minimum name length.

`default_max(`⟨*gender*⟩`)`
: For the given ⟨*gender*⟩, returns the culture's default maximum name length.

`generate(`⟨*gender*⟩`=None, `⟨*start*⟩`=None, `⟨*num*⟩`=None, `⟨*lmin*⟩`=None, `⟨*lmax*⟩`=None, `⟨*surnames*⟩`=True)`

Generates and returns a list of names for the given gender. If no gender is specified, it defaults to "`F`". The other parameters are:

⟨*start*⟩
: Specifies the required initial letter for the names to be generated. Does not apply to generated surname parts of the tuple if ⟨*surnames*⟩ is `True`.

⟨*num*⟩
: The number of names to generate. Defaults to 10.

⟨*lmin*⟩
: The minimum length of generated names.

⟨*lmax*⟩
: The maximum length of generated names.

⟨*surnames*⟩
: If this is `True`, the generated names will be tuples where the first name is a given name of the requested gender, and the second will be generated as a surname. Otherwise the generated list of names will be single string values.

Care is taken to avoid generating duplicate names, but only for the set of results of a given call to `generate()`.

Go Version

For the Go language version of this module, each culture is defined as a separate type which satisfies the `Culture` interface. This allows additional or custom cultures to be "plugged in" to the base package without altering the other code.

A map `namegen.Cultures` is provided which associates a human-readable culture name with a value of the corresponding type which a program may use where a `Culture` value is expected.

Given a culture, preference of gender, and a quantity of names to generate, calling `namegen.Generate` produces a slice of given names, or `namegen.GenerateWithSurnames` produces a slice of names, where each element is a slice of two strings (given name and surname).

See the documentation at pkg.go.dev for full details.

HISTORY

Written by Steve Willoughby, initially as a port of John Mechalas's original implemention in JavaScript.

BUGS

gma-pounds_to_grams(3)

See gma-weightsandmeasures(3), p. 351.

gma-preferences(3)

NAME

Preferences – Manage application preferences and initialization

SYNOPSIS

```
import MadScienceZone.GMA.Preferences
default_core_db_path() ⇒ ⟨path⟩

default_preferences_path() ⇒ ⟨path⟩

init_application(⟨config_path⟩=None, ⟨class_list⟩=None,
⟨open_db⟩=True) ⇒ ⟨preferences⟩

Preferences() ⇒ ⟨p⟩
Preferences.deserialize(⟨dict⟩) ⇒ ⟨p⟩
Preferences.dup(⟨other⟩) ⇒ ⟨p⟩
Preferences.from_file(⟨path⟩) ⇒ ⟨p⟩
⟨p⟩.all_networks
⟨p⟩.all_worlds
⟨p⟩.core_db
⟨p⟩.current_network_name
⟨p⟩.current_world_name
⟨p⟩.dark
⟨p⟩.db
⟨p⟩.network
⟨p⟩.scaling
⟨p⟩.world
⟨p⟩.open_database(⟨class_list⟩)
⟨p⟩.select_network(⟨net_name⟩)
⟨p⟩.select_world(⟨world_name⟩)
⟨p⟩.serialize() ⇒ ⟨dict⟩
⟨p⟩.deep_copy(⟨other⟩)
⟨p⟩.maphost() ⇒ (⟨hostname⟩, ⟨port⟩) or None
⟨p⟩.to_file(⟨path⟩)
⟨p⟩.update(⟨other⟩)
⟨p⟩.__eq__(⟨other⟩) ⇒ ⟨bool⟩

CasterData(⟨name⟩, ⟨cl⟩) ⇒ ⟨c⟩
CasterData.deserialize(⟨dict⟩) ⇒ ⟨c⟩
⟨c⟩.name
⟨c⟩.cl
⟨c⟩.serialize() ⇒ ⟨c⟩
```

```
⟨c⟩.__eq__(⟨other⟩) ⇒ ⟨bool⟩
⟨c⟩.__lt__(⟨other⟩) ⇒ ⟨bool⟩
CasterLevels() ⇒ ⟨cls⟩

InitiativeData(⟨name⟩, ⟨dex⟩, ⟨init_adj⟩, ⟨con⟩, ⟨hp⟩,
⟨blur_hp⟩, ⟨is_pc⟩=False, ⟨die_spec⟩=None) ⇒ ⟨i⟩
InitiativeData.deserialize(⟨dict⟩) ⇒ ⟨i⟩
⟨i⟩.name
⟨i⟩.dex
⟨i⟩.con
⟨i⟩.init_adj
⟨i⟩.hp
⟨i⟩.blur_hp
⟨i⟩.is_pc
⟨i⟩.die_spec
⟨i⟩.serialize() ⇒ ⟨dict⟩
⟨i⟩.__lt__(⟨other⟩) ⇒ ⟨bool⟩
⟨i⟩.__eq__(⟨other⟩) ⇒ ⟨bool⟩

InitiativeSeed() ⇒ ⟨is⟩

NetworkProfile(⟨hostname⟩=None, ⟨port⟩=2323) ⇒ ⟨n⟩
NetworkProfile.deserialize(⟨dict⟩) ⇒ ⟨n⟩
⟨n⟩.hostname
⟨n⟩.port
⟨n⟩.serialize() ⇒ ⟨dict⟩
⟨n⟩.__eq__(⟨other⟩) ⇒ ⟨bool⟩

World(⟨calendar_type⟩=golarion, ⟨module⟩=None,
⟨blur_hp⟩=None, ⟨db_name⟩=None, ⟨display_name⟩=None,
⟨password⟩=None, ⟨initiative_backup_path⟩=None) ⇒ ⟨w⟩
World.deserialize(⟨dict⟩) ⇒ ⟨w⟩
⟨w⟩.blur_hp
⟨w⟩.calendar_type
⟨w⟩.caster_levels
⟨w⟩.db_name
⟨w⟩.display_name
⟨w⟩.initiative_seed
⟨w⟩.module
⟨w⟩.password
⟨w⟩.initiative_backup_path
⟨w⟩.serialize() ⇒ ⟨dict⟩
⟨w⟩.__eq__(⟨other⟩) ⇒ ⟨bool⟩

ProfileDataError
ProfileDataError.NoSuchNetwork
ProfileDataError.NoSuchWorld
```

DESCRIPTION

The GMA toolsuite uses a common, centralized preferences data file (with the exception of the mapper client, which has its own). As version 7 introduced a core database to hold SRD and local campaign information, which requires a standard setup sequence including the opening of that database, this module includes an `init_application` function which makes it easier for all GMA-related programs to perform this standard setup.

The information in the GMA preferences file is available via the ⟨*Preferences*⟩ class as documented below. (The preferences system was then introduced early, in version 6.15, but in this case the references to the core database are not used by GMA.)

FUNCTIONS

`default_core_db_path()` ⇒ ⟨*path*⟩

Returns the default pathname for the GMA SRD core database file.

`default_preferences_path()` ⇒ ⟨*path*⟩

Returns the default pathname for the GMA preferences file.

`init_application(`⟨*config_path*⟩`,` ⟨*class_list*⟩`,` ⟨*open_db*⟩`)` ⇒ ⟨*preferences*⟩

This performs the essential startup functions for any GMA application. It reads the preferences data from the named ⟨*config_path*⟩, or from the default path if that parameter is not provided or is `None`. If ⟨*open_db*⟩ is `True`, it will also open the core database and register it with the list of GMA object classes passed in ⟨*class_list*⟩. This open database's connection handle is available in ⟨*preferences*⟩`.db`.

OBJECTS

CasterData

Each instance of this class describes a creature (usually a PC) whose spellcasting stats are important enough to keep available at all times. It has the same `serialize()` and `deserialize()` methods as the other classes in this module, and allows for comparison (where ⟨*a*⟩<⟨*b*⟩ compares ⟨*a*⟩`.name` and ⟨*b*⟩`.name`) as well as tests for equality (where all elements must be equal). The attributes include:

`name`	The creature's name.
`cl`	Their caster level.

CasterLevels

This class works exactly as a `list` object does. It holds the list of ⟨*CasterData*⟩ values which are in play (usually the PC party spellcasters).

InitiativeData

Each instance of this class describes a creature (usually a PC) whose combat stats are important enough to keep available at all times. It has the same `serialize()` and `deserialize()` methods as the other classes in this module.

`InitiativeData` supports comparison (where ⟨*a*⟩<⟨*b*⟩ means ⟨*a*⟩`.name`<⟨*b*⟩`.name`) and equality (where all attributes must be equal for the objects to be equal).

The attributes include:

`name`	The creature's name.
`con`	Constitution score.
`dex`	Dexterity score.
`init_adj`	The initiative modifier value *not including* the bonus supplied by their Dexterity score.
`hp`	Maximum hit points when fully healed.
`blur_hp`	The percentage of hit point blurring to use for this individual.
`is_pc`	If `True`, this creature is a player character; otherwise it is an NPC or monster.
`die_spec`	If empty, assume we roll 1d20 to determine initiative order for this creature. Otherwise, this string gives the appropriate die roll (not including the standard modifiers GMA already adds to the 1d20 to roll initiative). For example, the value "`d20 best of 2`" means to roll the d20 twice, taking the better roll, along with the usual modifiers.

InitiativeSeed

This class works exactly as a `list` object does. It holds the list of ⟨*InitiativeData*⟩ values which are in play (usually the PC party members).

NetworkProfile

Describes each network connection profile defined in the preferences. It has the standard `serialize()` and `deserialize()` methods as other classes in this module, and supports comparison for equality (where all attributes must be equal). It also has these attributes:

`hostname`	The name of the server to connect to. If `None`, do not connect to any server.
`port`	The TCP port of the server.

Preferences

The `Preferences` class encapsulates the GMA preferences for use by the application. It includes the following public attributes:

`all_networks`	A list of all network profiles defined in the preferences data. This is a dictionary mapping the name of a network profile to a `NetworkProfile` object describing it.
`all_worlds`	A list of all world profiles defined in the preferences data. This is a dictionary mapping the name of a campaign world profile to a `World` object describing it.
`core_db`	The pathname of the core SRD database. (Used in GMA version 7+ only.)
`current_network_name`	The name of the current network profile selected by the user. This must be one of the keys in `all_networks` or `None` if no network profile is selected at all.
`current_world_name`	The name of the current campaign world profile selected by the user. This must be one of the keys in `all_worlds` or `None` if no world profile is selected at all.
`dark`	This boolean value is `True` if the user selected a dark color theme.
`db`	A `sqlite3.Connection` object which corresponds to the open SRD database, or `None` if no database is open. (Used in GMA version 7+ only.)
`network`	A reference to the `NetworkProfile` object of the current network profile or `None` if none is selected.
`scaling`	The scaling factor for displayed GUI widgets. Normally this should be `1.0`, but may be increased to `2.0`, etc., to adjust the look of the application appropriately on high-resolution displays.
`world`	A reference to the `World` object of the current campaign world or `None` if none is selected.

The following class methods are available:

`Preferences()`
: The default constructor takes no arguments and returns a default `Preferences` object.

`Preferences.deserialize(`⟨*dict*⟩`)`
: Given a dictionary such as that output by the `serialize()` method, this constructs and returns a new `Preferences` object with values from the provided ⟨*dict*⟩.

`Preferences.from_file(`⟨*path*⟩`)`
: Loads JSON-format preferences data from the named ⟨*path*⟩ and returns a new `Preferences` object with the values read from that file.

The following methods are avaialble for objects of this type:

⟨*p*⟩`.open_database(`⟨*class_list*⟩`)`
: Opens the database configured as the ⟨*p*⟩`.core_db` value. The open connection object is available as the ⟨*p*⟩`.db` attribute. This database connection is then registered with each of the classes in ⟨*class_list*⟩ by invoking the `register_db()` class method in each.

⟨*p*⟩`.select_network(`⟨*net_name*⟩`)`
: Make the network profile named ⟨*net_name*⟩ the currently-selected one. Raises a `NoSuchNetwork` exception if there is no profile with that name.

⟨*p*⟩`.select_world(`⟨*world_name*⟩`)`
: Make the campaign world profile named ⟨*world_name*⟩ the currently-selected one. Raises a `NoSuchWorld` exception if there is no profile with that name.

⟨*p*⟩`.serialize()`
: Returns a dictionary with all the information represented by the `Preferences` object ⟨*p*⟩, suitable for encoding (say, as a JSON string) and saving for later use.

⟨*p*⟩`.to_file(`⟨*path*⟩`)`
: Serialize the contents of ⟨*p*⟩ and write them in JSON format to the named disk file.

⟨*p*⟩`.update(`⟨*other*⟩`)`
: Make a shallow copy of the attributes of the ⟨*other*⟩ `Preferences` object into the receiver ⟨*p*⟩.

World

Objects of this class describe the campaign world in play. It has the same `serialize()` and `deserialize()` methods as are standard for the classes in this module. It supports comparisons for equality (where all attributes must be equal). Its attributes include:

`blur_hp`	The percentage of "blur" to add to obscure hit points reported out to mapper clients, or `None`.
`calendar_type`	The calendar system in use. Must be one of the strings accepted by `GameCalendar.select_calendar()`.
`caster_levels`	A `CasterLevels` object holding all of the PC spellcasters' levels.
`db_name`	The name of the SRD database file or `None`.
`display_name`	The displayed name of the campaign or `None`.
`initiative_seed`	An `InitiativeSeed` object holding all the PC stats that should always be included in initiative.
`module`	The pathname of the module encounter (.gmo) file or `None` if there is none.
`password`	The password with which to log in to the GMA server or
`initiative_backup_path`	The pathname where GMA should occasionally write backup copies of the current initiative list. If no backup copies should be made, this has the value `None`.

SEE ALSO

`gma-gamecalendar`(3), `gma`(6), `gma-preferences`(6).

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

This functionality was created for GMA version 7.0. It was pulled back into version 6.15 so that it could be available ahead of the 7.0 release.

BUGS

The `CasterData` class doesn't account for spellcasters with multiple caster levels.

COPYRGHT

Part of the GMA software suite, copyright © 1992–2025 by Steven L. Willoughby, Aloha, Oregon, USA. All Rights Reserved. Distributed under BSD-3-Clause License.

gma-ps_str(3)

See gma-markuptext(3), p. 298.

gma-ps_str_raw(3)

See gma-markuptext(3), p. 298.

gma-psformformatter(3)

See gma-markuptext(3), p. 298.

gma-read_map_file(3)

See gma-mapobject(3), p. 274.

gma-read_map_meta_data(3)

See gma-mapobject(3), p. 274.

gma-romannumerals(3)

NAME

RomanNumerals – Support for Roman numeral values

SYNOPSIS

```
import MadScienceZone.Common.RomanNumerals

⟨r⟩ = to_roman(⟨i⟩)

⟨i⟩ = from_roman(⟨r⟩)
```

DESCRIPTION

This module contains two functions for converting values between integers and Roman numerals.

The `to_roman()` function accepts an integer value ⟨*i*⟩ and returns a string containing its representation in Roman numerals. If ⟨*i*⟩=0, the string `0` is returned. If ⟨*i*⟩<0, an empty string is returned. The highest Roman numeral supported is M, so numbers much larger than 1000 may have a lot of Ms at the start of the string.

The `from_roman()` function accepts a string value ⟨*r*⟩ which is assumed to be a value expressed in Roman numerals. It will return the corresponding integer value. A string containing only a zero is considered to be the value 0. Otherwise the string must consist only of the letters I, V, X, L, C, D, and M. A `ValueError` exception is raised if the string cannot be understood. The input string may be in any case.

SEE ALSO

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

gma-save_map_file(3)

See gma-mapobject(3), p. 274.

gma-simpleconfigfile(3)

NAME

SimpleConfigFile – Support for simplified configuration file reading

SYNOPSIS

```
import MadScienceZone.Common.SimpleConfigFile

⟨cfg⟩ = SimpleConfigFile(⟨filename⟩)
⟨value⟩ = ⟨cfg⟩.get(⟨key⟩)
⟨value⟩ = ⟨cfg⟩.getint(⟨key⟩)
⟨cfg⟩.store(⟨key⟩, ⟨value⟩)
```

DESCRIPTION

Normally, we would use a more featured configuration file parser such as those built into the Python standard library. However, to enable our programs to correctly read the simpler file format described below, this class is provided. For example, the GMA `gma-mapper`(6) program reads a configuration file of this format without support for the more sophisticated features of the other formats.

A `SimpleConfigFile` object is constructed with a ⟨*filename*⟩ which will be opened, read into memory, and closed at once. After that, the following methods may be called to find values in this file.

⟨*value*⟩=⟨*cfg*⟩`.get(`⟨*key*⟩`)`
: If the named ⟨*key*⟩ exists in the file, return its value. Otherwise return `None`.

⟨*value*⟩=⟨*cfg*⟩`.getint(`⟨*key*⟩`)`
: Just like the `get()` method, but returns its value as an integer instead of a string. Returns `None` if there is no such ⟨*key*⟩ but raises an error if the value stored for that ⟨*key*⟩ cannot be understood as an integer.

⟨*cfg*⟩`.store(`⟨*key*⟩`,` ⟨*value*⟩`)`
: Store a new ⟨*value*⟩ for ⟨*key*⟩ in the configuration object ⟨*cfg*⟩. If that key already existed, it will be replaced. Otherwise a new entry will be created for it. Note that this is for the internal use of the application only; no support is currently provided for writing this data back to a file.

FILE FORMAT

This simple file format is simply a set of lines, each of which has the form

⟨*key*⟩=⟨*value*⟩

or

⟨*key*⟩

There is no support for continuing long lines or for defining multi-line values. Any amount of whitespace may appear on either side of the equals sign or at the beginning or end of the line, all of which will be discarded.

Everything in the file is case-sensitive.

In the second form (no value), the ⟨*key*⟩ is given a value of `True`.

Blank lines and lines beginning with a # sign are ignored.

SEE ALSO

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

gma-socketinterface(3)

NAME

SocketInterface – Interface to GMA mapper communication service

SYNOPSIS

```
import MadScienceZone.GMA.SocketInterface

⟨pv⟩=PackageVersion(⟨os⟩, ⟨arch⟩, ⟨version⟩, ⟨token⟩)

⟨char⟩=MapCharacter(⟨name⟩, ⟨mapid⟩, ⟨color⟩, ⟨size⟩,
⟨skin_sizes⟩=None, ⟨skin⟩=0, ⟨poly_gm⟩=False)

⟨cond⟩=MapCondition(⟨name⟩, ⟨shape⟩, ⟨color⟩, ⟨description⟩="")

⟨peer⟩=Peer(⟨addr⟩, ⟨user⟩, ⟨client⟩, ⟨last_polo⟩=0,
⟨is_auth⟩=False, ⟨is_me⟩=False)
("you"|"peer", ⟨addr⟩, ⟨user⟩, ⟨client⟩, ⟨is_auth⟩, ⟨last_polo⟩) =
⟨peer⟩.to_tuple()

⟨imgdef⟩=ImageInstance(⟨zoom⟩, ⟨file⟩, ⟨image_data⟩=None,
⟨is_local_file⟩=False)

⟨size⟩=ImageInstance(⟨zoom⟩, ⟨file⟩=None, ⟨image_data⟩=None,
⟨is_local_file⟩=False)
⟨img⟩=ImageDefinition(⟨name⟩, ⟨sizes⟩=None)
⟨img⟩.add_size(⟨size⟩)

⟨slot⟩=MapInitiativeSlot(⟨name⟩, ⟨is_holding⟩=False,
⟨has_readied_action⟩=False, ⟨hp⟩=0, ⟨is_flat_footed⟩=False,
⟨slot_no⟩=0)

⟨s⟩=SocketInterface(⟨maphost⟩, ⟨authenticator⟩=None,
⟨initial_commands⟩=None, ⟨retries⟩=1, ⟨diag_callback⟩=None,
⟨sleep_callback⟩=None, ⟨protocol_debug⟩=None,
⟨subscribe_to⟩=None, ⟨pass_callback⟩=None, ⟨use_tk⟩=False,
⟨preset_details=False)⟩
⟨s⟩.dispatch(⟨handler⟩, ⟨verbose⟩=False, ⟨timeout⟩=0)
⟨s⟩.add_image(⟨img_def⟩)
⟨s⟩.add_obj_attributes(⟨obj_id⟩, ⟨attrname⟩, ⟨values⟩)
⟨s⟩.adjust_view(⟨x⟩, ⟨y⟩, ⟨grid_label⟩=None)
⟨s⟩.allow(⟨dice_color_boxes⟩=False, ⟨dice_color_labels⟩=False,
⟨gma_markup⟩=False)
⟨s⟩.cache_file(⟨server_id⟩)
```

⟨*s*⟩.chat_message(⟨*message*⟩, ⟨*sender*⟩=None, ⟨*recipients*⟩=None, ⟨*to_all*⟩=False, ⟨*to_GM*⟩=False, ⟨*message_id*⟩=None, ⟨*sent*⟩=None, ⟨*replay*⟩=False, ⟨*origin*⟩=False, ⟨*markup*⟩=False)
⟨*package_version*⟩= ⟨*s*⟩.check_version_of(⟨*program*⟩, ⟨*current_version*⟩, ⟨*target_os*⟩, ⟨*target_arch*⟩)
⟨*s*⟩.clear(⟨*obj_id*⟩)
⟨*s*⟩.clear_chat(⟨*by*⟩=None, ⟨*silent*⟩=False, ⟨*target*⟩=None, ⟨*message_id*⟩=None)
⟨*s*⟩.clear_from(⟨*server_id*⟩, ⟨*is_local_file*⟩=False)
⟨*s*⟩.combat_mode(⟨*enabled*⟩)
⟨*s*⟩.comment(⟨*text*⟩)
⟨*s*⟩.define_dice_presets(⟨*preset_list*⟩, ⟨*append*⟩=False, ⟨*for_user*⟩=None, ⟨*globals_only*⟩=False)
⟨*s*⟩.echo(⟨*s*⟩="", ⟨*i*⟩=0, ⟨*b*⟩=False, ⟨*o*⟩=None, ⟨*received_time*⟩=None, ⟨*sent_time*⟩=None)
⟨*s*⟩.failed(⟨*reason*⟩, ⟨*is_error*⟩=False, ⟨*is_discretionary*⟩=False, ⟨*command*⟩="", ⟨*request_id*⟩="" ⟨*requested_by*⟩=None, ⟨*requesting_client*⟩=None)
⟨*s*⟩.filter_core_data(⟨*item_type*⟩, ⟨*code*⟩="", ⟨*name*⟩="", ⟨*request_id*⟩="",
⟨*s*⟩.hit_point_acknowledge(⟨*request_id*⟩="", ⟨*requested_by*⟩=None, ⟨*requesting_client*⟩=None)
⟨*s*⟩.hit_point_request(⟨*request_id*⟩="", ⟨*description*⟩="", ⟨*targets*⟩=None, ⟨*health*⟩=None, ⟨*tmp_hp*⟩=None, ⟨*requested_by*⟩=None, ⟨*requesting_client*⟩=None)
⟨*s*⟩.update_progress(⟨*operation_id*⟩, ⟨*title*⟩, ⟨*value*⟩, ⟨*max_value*⟩, ⟨*is_done*⟩=False, ⟨*is_timer*⟩=False⟨,⟩ ⟨*targets*⟩=None)
⟨*s*⟩.filter_dice_presets(⟨*regex*⟩, ⟨*for_user*⟩=None, ⟨*globals_only*⟩=False)
⟨*s*⟩.load_from(⟨*server_id*⟩, ⟨*is_local_file*⟩=False, ⟨*cache_only*⟩=False, ⟨*merge*⟩=False)
⟨*s*⟩.marco()
⟨*s*⟩.mark(⟨*x*⟩, ⟨*y*⟩)
⟨*s*⟩.place_someone(⟨*obj_id*⟩, ⟨*color*⟩, ⟨*name*⟩, ⟨*size*⟩, ⟨*obj_type*⟩, ⟨*gx*⟩, ⟨*gy*⟩, ⟨*reach*⟩=0, ⟨*health*⟩=None, ⟨*skin*⟩=0, ⟨*skin_sizes*⟩=None, ⟨*elevation*⟩=0, ⟨*note*⟩=None, ⟨*status_list*⟩=None, ⟨*area_of_effect*⟩=None, ⟨*move_mode*⟩=0, ⟨*killed*⟩=False, ⟨*dim*⟩=True, ⟨*hidden*⟩=False, ⟨*disp_size*⟩=None, ⟨*poly_gm*⟩=False)
⟨*s*⟩.polo()
⟨*s*⟩.query_dice_presets(⟨*for_user*⟩=None, ⟨*globals_only*⟩=False)
⟨*s*⟩.query_image(⟨*image_def*⟩)
⟨*s*⟩.query_peers()
⟨*s*⟩.remove_obj_attributes(⟨*obj_id*⟩, ⟨*attrname*⟩, ⟨*values*⟩)
⟨*s*⟩.roll_dice(⟨*roll_spec*⟩, ⟨*recipients*⟩=None, ⟨*to_all*⟩=False, ⟨*blind_to_gm*⟩=False, ⟨*request_id*⟩=None)
⟨*s*⟩.roll_result(⟨*title*⟩, ⟨*result*⟩, ⟨*details*⟩, ⟨*sender*⟩, ⟨*recipients*⟩=None, ⟨*to_all*⟩=False, ⟨*blind_to_gm*⟩=False, ⟨*message_id*⟩=None, ⟨*more_results*⟩=False, ⟨*request_id*⟩=None, ⟨*invalid_request*⟩=False, ⟨*result_suppressed*⟩=False, ⟨*sent*⟩=None, ⟨*replay*⟩=False, ⟨*origin*⟩=False)
⟨*s*⟩.subscribe(⟨*msg_list*⟩)
⟨*s*⟩.sync()
⟨*s*⟩.sync_chat(⟨*target*⟩=None)
⟨*s*⟩.toolbar(⟨*enabled*⟩)
⟨*s*⟩.timer_acknowledge(⟨*request_id*⟩="", ⟨*requested_by*⟩=None,

```
⟨s⟩.timer_request(⟨request_id⟩="", ⟨description⟩="",
⟨expires⟩="", ⟨targets⟩=None, ⟨show_to_all⟩=True,
⟨is_running⟩=True, ⟨requested_by⟩=None, ⟨requesting_client⟩=None)
⟨s⟩.update_clock(⟨abs⟩, ⟨rel⟩, ⟨running⟩=False)
⟨s⟩.update_dice_presets(⟨preset_list⟩, ⟨for_user⟩=None,
⟨delegates⟩=None, ⟨delegate_for⟩=None, ⟨globals_only⟩=False,
⟨global_presets⟩=None)
⟨s⟩.update_initiative_list(⟨slot_list⟩)
⟨s⟩.update_obj_attributes(⟨obj_id⟩, ⟨kvdict⟩)
⟨s⟩.update_peer_list(⟨peer_list⟩)
⟨s⟩.update_status_marker(⟨condition⟩, ⟨shape⟩, ⟨color⟩,
⟨description⟩="")
⟨s⟩.update_turn(⟨actor_id⟩, ⟨rounds⟩, ⟨hours⟩, ⟨minutes⟩,
⟨seconds⟩, ⟨count⟩)
⟨s⟩.protocol_debug(⟨handle⟩=None)
⟨s⟩.hold()
⟨s⟩.release()
⟨s⟩.sendall(⟨data⟩)
⟨data⟩=⟨s⟩.recv(⟨size⟩)
⟨s⟩.reconnect()
⟨data⟩=⟨s⟩.read_poll(⟨verbose⟩=False, ⟨timeout⟩=0)
⟨data⟩=⟨s⟩.parse_data_packet(⟨raw_line⟩)
⟨s⟩.dispatch( ⟨handler⟩, ⟨verbose⟩=False, ⟨timeout⟩=0)
SocketInterface.all_messages

⟨h⟩=CreatureHealth(⟨max_hp⟩, ⟨lethal_damage⟩=0,
⟨nonlethal_damage⟩=0, ⟨con⟩=0, ⟨is_flat_footed⟩=False,
⟨is_stable⟩=False, ⟨condition⟩=None, ⟨hp_blur⟩=0, ⟨tmp_hp⟩=0,
⟨tmp_damage⟩=0)
⟨h⟩.to_dict()

⟨hr⟩=HitPointHealthRequest(⟨max_hp⟩=0, ⟨lethal_damage⟩=0,
⟨nonlethal_damage⟩=0)
⟨hr⟩=HitPointHealthRequest.from_dict(⟨dict⟩)
⟨hr⟩.to_dict()

⟨tr⟩=HitPointTmpHPRequest(⟨tmp_hp⟩=0, ⟨tmp_damage⟩=0,
⟨expires⟩=None)
⟨tr⟩=HitPointTmpHPRequest.from_dict(⟨dict⟩)
⟨tr⟩.to_dict()
```

DESCRIPTION

The `SocketInterface` class handles the details of communicating with the GMA mapper service communication channel used to keep the mapper clients in sync with each other and with the other GMA tools.

When instantiated, a `SocketInterface` object ⟨*s*⟩ establishes a connection to the specified host, logs in, and then becomes the point of contact to interact with the mapper service. The constructor and methods available are described below.

⟨*s*⟩=`SocketInterface(`⟨*maphost*⟩`,` ⟨*authenticator*⟩=`None,`
⟨*initial_commands*⟩=`None,` ⟨*retries*⟩=`1,` ⟨*diag_callback*⟩=`None,`
⟨*sleep_callback*⟩=`None,` ⟨*protocol_debug*⟩=`None,` ⟨*subscribe_to*⟩=`None,`
⟨*pass_callback*⟩=`None,` ⟨*use_tk*⟩=`False)`

Establishes a connection to the service at ⟨*maphost*⟩ (which is a tuple of (⟨*hostname_or_IP*⟩, ⟨*port*⟩)). If the connection fails, it will try again until the number of ⟨*retries*⟩ specified has been exhausted. It will sleep for 1 second between attempts. Once the connection is made, if the server requires authentication and an ⟨*authenticator*⟩ object is given, that object will be used to negotiate authentication with the server. This should be an instance of a `Mapper.Authentication.Authenticator` object. If the password stored in ⟨*authenticator*⟩ is a single question mark ("?"), then the user is prompted for their actual password, which is then inserted into the authenticator object before using it. The server's initial set of configuration data and/or commands are received and stored in the new `SocketInterface` object as appropriate.

If ⟨*initial_commands*⟩ is provided, the commands it contains are sent to the server immediately.

If ⟨*diag_callback*⟩ is given, then all diagnostic messages will be sent to that function as the sole parameter. Otherwise, in some cases an attempt will be made to alert the user using a Tk dialog box, but if that fails (such as, the application does not have a Tk interface running), it will fall back to printing diagnostic messages to the standard output.

If ⟨*sleep_callback*⟩ is given, then any delays needed by the object will be performed by calling the provided function with two parameters: number of seconds to delay, and a callable function which should be invoked when that delay has expired. Otherwise, `sleep()` will be called.

If ⟨*subscribe_to*⟩ is specified, its value will be passed to the `subscribe()` method to limit the set of messages the server will send to this client.

If debugging a client, you can pass an open file-like object to the ⟨*protocol_debug*⟩ parameter. Extra debugging information will be written to that file.

If ⟨*pass_callback*⟩ is given, then any time the socket interface needs to prompt the user for a password, it will invoke the function passed as the value to this parameter, with two arguments: a title and a prompt. The value returned by this callback function should be the user's password. If

this is not given, at attempt will be made to prompt for a password using a Tk dialog box, and if that is unsuccessful, the user will be prompted to type it on the standard input.

If ⟨*use_tk*⟩ has a true value, the Tk features will be attempted; otherwise they never will.

The following attributes are available after the object is constructed:

⟨*s*⟩`.protocol`
: The protocol version number used by the server.

⟨*s*⟩`.characters`
: A dictionary mapping character names to corresponding `MapCharacter` objects describing the main party (what would be on the quick placement list of the Mapper tool's pop-up context menu).

⟨*s*⟩`.conditions`
: A dictionary mapping the name of a condition to a corresponding `MapCondition` object describing how a particular condition is to be drawn on a map display.

⟨*s*⟩`.subscribed_messages`
: The set of all server messages this client is currently subscribed to receive.

⟨*s*⟩`.preamble`
: The initial greeting sent by the server.

⟨*s*⟩`.username`
: The name of the logged-in user.

⟨*s*⟩`.package_updates_available`
: A dictionary mapping package names to a list of `PackageVersion` objects for the advertised versions of each package available for download.

⟨*s*⟩`.calendar`
: The name of the calendar system in play.

High-Level Interface

Once a `SocketInterface` object ⟨*s*⟩ is constructed, it should be logged in to a server and ready to exchange commands with it. The following methods expose the API to the caller to interact with the map service:

⟨*s*⟩`.protocol_debug(`⟨*handle*⟩`=None)`
: You can call this to turn on or off debugging on the fly. It performs just as the ⟨*protocol_debug*⟩ parameter to the constructor.

⟨*s*⟩`.dispatch(`⟨*handler*⟩`,` ⟨*verbose*⟩`=False,` ⟨*timeout*⟩`=0)`
: This method is analogous to the lower-level `read_poll()` method (and the ⟨*verbose*⟩ and ⟨*timeout*⟩ parameters are handled identically), but rather than simply returning the data sent by the server as a string, it interprets the data according to the mapper protocol and invokes the corresponding handler method in your ⟨*handler*⟩ object.

  To make one of these, create your own handler subclass derived from `MapperProtocolHandler`. The base class is full of stub methods which ignore the incoming server messages. In your own class, simply override all of the ones you care to receive in your client.

  There will be a method corresponding to each of the following methods used to send data to the server, with identical calling semantics. Thus, for example, given that you can call ⟨*s*⟩`.toolbar(`⟨*enabled*⟩`)` to order peers to turn on or off their toolbars, if you receive the toolbar command from the server, `dispatch` will invoke the method ⟨*handler*⟩`.toolbar(`⟨*enabled*⟩`)` for your ⟨*handler*⟩ object to respond in whatever way you wish.

  In addition to the protocol methods, two additional methods are defined:

  ⟨*handler*⟩`.unknown(`⟨*command*⟩`,` ⟨*args*⟩`)`
  : The server sent a command that we don't recognize. The command (as a string) and its arguments (as a list of strings) are passed as parameters. The default behavior is to ignore this event.

  ⟨*handler*⟩`.error(`⟨*description*⟩`,` ⟨*data*⟩`)`
  : An error was encountered as described in the ⟨*description*⟩ parameter while trying to process the incoming server data. The raw data string as received is passed as ⟨*data*⟩. The default behavior is to ignore these errors.

⟨*s*⟩`.add_image(`⟨*img_def*⟩`)`
: Define a graphic image for use by the mapper clients.

⟨*s*⟩`.add_obj_attributes(`⟨*obj_id*⟩`,` ⟨*attrname*⟩`,` ⟨*values*⟩`)`
: For an existing map element or creature with the given ⟨*obj_id*⟩, which has an attribute called ⟨*attrname*⟩ whose value is a list of values that can be present or not (a good example of which is the `STATUSLIST` attribute of creatures), this command orders clients to update that object to add the values in the ⟨*values*⟩ list to the existing set of values in that attribute. This is like a set union operation; if any of the values is already in that attribute they are not added twice.

⟨*s*⟩`.adjust_view(`⟨*x*⟩`,` ⟨*y*⟩`,` ⟨*grid_label*⟩`=None)`
: Tell other clients they should adjust their map displays such that the upper-left of the displayed area shows the grid square with the given

⟨*grid_label*⟩. If the ⟨*grid_label*⟩ is missing, empty, or the client is unable to properly interpret it, then the client should adjust their map display to ⟨*x*⟩ in the horizontal direction (as a fraction of the full width) and ⟨*y*⟩ in the vertical direction (as a fraction of the full height). Both values should be in the range 0.0 to 1.0.

⟨*s*⟩`.allow(`⟨*dice_color_boxes*⟩`=False, `⟨*dice_color_labels*⟩`=False, `⟨*gma_markup*⟩`=False)`

This tells the server to allow the use of optional features that this client supports. Currently this includes the following features:

⟨*dice_color_boxes*⟩
: If `True`, the client allows special formatting characters in die-roll title strings.

⟨*dice_color_labels*⟩
: If `True`, this enables the use of colorized labels in the die-roll result sets.

⟨*gma_markup*⟩
: if `True`, this signals that the client is capable of interpreting GMA markup formatting codes in chat messages.

⟨*s*⟩`.cache_file(`⟨*server_id*⟩`)`
: Tell clients that a map file with the given ⟨*server_id*⟩ will be interesting in the future. They may wish to load that file from the server to their cache in anticipation of being told to load its contents later.

⟨*s*⟩`.chat_message(`⟨*message*⟩`, `⟨*sender*⟩`=None, `⟨*recipients*⟩`=None, `⟨*to_all*⟩`=False, `⟨*to_gm*⟩`=False, `⟨*message_id*⟩`=None, `⟨*sent*⟩`=None, `⟨*replay*⟩`=False, `⟨*origin*⟩`=False, `⟨*markup*⟩`=False)` Send ⟨*message*⟩ to some other peers. If no recipients are listed, the message will be sent back to you alone. Otherwise, if ⟨*to_all*⟩ is `True`, then it is transmitted to all peers. Otherwise, it is sent to the list of peers in the ⟨*recipients*⟩ list. The ⟨*sender*⟩ and ⟨*message_id*⟩ parameters are ignored (your logged-in username is always used regardless of what is passed for ⟨*sender*⟩) but is included here for symmetry with the ⟨*handler*⟩`.chat_message()` method where you will receive incoming chat messages. In that case you'll read the sender's name in that parameter. If ⟨*to_GM*⟩ is true, the message will be sent privately to the GM. If you wish to include markup codes in the message, set the ⟨*markup*⟩ parameter to `True` to notify recipients that they should interpret the message in that context (also, with that flag set, the server will automatically scrub those codes out of the message for clients which have not indicated that they are capable of understanding those formatting codes at all). If you include GMA formatting codes in your message, set the ⟨*markup*⟩ value to `True`. When receiving a chat message in your handler, the server will fill in the ⟨*message_id*⟩ and ⟨*sent*⟩ fields to with a unique ID and the time

the message was sent. It will also set ⟨*replay*⟩ if the message is being resent again as opposed to being sent for the first time during the original conversation. It will set the ⟨*origin*⟩ flag only when sending the message back to the client who originated the message as opposed to other clients receiving their copies of it.

⟨*s*⟩`.check_version_of(`⟨*program*⟩`,` ⟨*current_version*⟩`,` ⟨*target_os*⟩`=None,` ⟨*target_arch*⟩`=None)`

Check to see if the server advertised a version of the specified ⟨*program*⟩ that is newer than ⟨*current_version*⟩. If you want to see if there's a version of a particular OS or architecture, specify their names as well. If one is found, a `PackageVersion` object describing it will be returned. Otherwise this method returns `None`.

⟨*s*⟩`.clear(`⟨*obj_id*⟩`)`

Remove the object with the given ⟨*obj_id*⟩ from the map. This may be the unique ID assigned to the object, the name of a creature as displayed on the map, or the wildcards `*` for all objects, `E*` for all map elements, `M*` for all monster tokens, or `P*` for all player tokens.

⟨*s*⟩`.clear_chat(`⟨*by*⟩`=None,` ⟨*silent*⟩`=False,` ⟨*target*⟩`=None,` ⟨*message_id*⟩`=None)`

Clear the server's chat history. The user who requested this action is named in the ⟨*by*⟩ parameter (which may be `None` if we don't know who it was for some reason). If ⟨*silent*⟩ is true, the clear should be done without making an announcement about it. If ⟨*target*⟩ is an integer value, it controls which messages are cleared with the same meaning as the same parameter for the `sync_chat()` method. The ⟨*message_id*⟩ is ignored (but retained here for consistency with the handler's method). Received events will have a message ID assigned by the server as chat messages and die roll results do.

⟨*s*⟩`.clear_from(`⟨*server_id*⟩`,` ⟨*is_local_file*⟩`=False)`

Functionally analogous to the `load_from()` method, in that it asks clients to load the map file with the given ⟨*server_id*⟩, but rather than adding the objects from that file into the map, the client should ⟨*remove*⟩ all of the objects from its map. It is not an error if they weren't loaded.

⟨*s*⟩`.combat_mode(`⟨*enabled*⟩`)`

Order clients to go into combat mode if ⟨*enabled*⟩ is `True`, otherwise they should revert to normal mode.

⟨*s*⟩`.comment(`⟨*text*⟩`)`

Send a comment ⟨*text*⟩ to the server. This provides informational context to anyone observing the data flow, perhaps to aid in debugging. The ⟨*text*⟩ parameter should be a simple string of characters without newlines.

⟨*s*⟩`.define_dice_presets(`⟨*preset_list*⟩`,` ⟨*append*⟩`=False,` ⟨*for_user*⟩`=None,` ⟨*globals_only*⟩`=False)`

Defines your set of die-roll presets to that described by ⟨*preset_list*⟩, replacing all previously-defined presets, unless ⟨*append*⟩ is `True`, in which case it appends ⟨*preset_list*⟩ to the existing ones. The ⟨*preset_list*⟩ parameter is a list of presets, where each element in the list is a `DieRollPreset` or `DieRollObject` object. If ⟨*for_user*⟩ is not `None` or the empty string, it names the user for whom to set the presets. Otherwise it sets the logged-in user's presets. Only the GM or authorized delegate may alter the presets for another user. If ⟨*globals_only*⟩ is true, this defines or appends to the set of system-wide global presets instead of an individual user's.

⟨*s*⟩`.echo(`⟨*s*⟩`="",` ⟨*i*⟩`=0,` ⟨*b*⟩`=False,` ⟨*o*⟩`=None,` ⟨*received_time*⟩`=None,` ⟨*sent_time*⟩`=None)`

When the server receives an echo message from you, it will send it back to you again. This may be useful for synchronizing actions between client and server, given the asynchronous nature of the communications protocol. The parameters allow for sending arbitrary data with the echo message. For convenience, they are typed as string, integer, boolean, and dict, respectively, to make it simple to pass values.

Never use the ⟨*received_time*⟩ or ⟨*sent_time*⟩ parameters when initiating an echo request to the server. However, these values will be filled in by the server for its response to you, and will be visible to your incoming echo message handler.

⟨*s*⟩`.filter_dice_presets(`⟨*regex*⟩`,` ⟨*for_user*⟩`=None,` ⟨*globals_only*⟩`=False)`

Deletes all die-roll presets whose names match the given regular expression ⟨*regex*⟩. If ⟨*globals_only*⟩ is true, items are filtered out of the system-wide list instead of an individual user's.

⟨*s*⟩`.hit_point_acknowledge(`⟨*request_id*⟩`="",` ⟨*requested_by*⟩`=None,` ⟨*requesting_client*⟩`=None)`

Acknowledge that the specified request for hit points was accepted. This can only be issued by the GM.

⟨*s*⟩`.hit_point_request(`⟨*request_id*⟩`="",` ⟨*description*⟩`="",` ⟨*targets*⟩`=None,`
⟨*health*⟩`=None,` ⟨*tmp_hp*⟩`=None,` ⟨*requested_by*⟩`=None,` ⟨*requesting_client*⟩`=None)`
Request a change to the target creature(s)' permanent hit point totals (if ⟨*health*⟩ is provided as a `HitPointHealthRequest` object), in which case ⟨*targets*⟩ should normally be a list with a single element naming the PC to be affected; or request that a number of temporary hit points be allocated to the creatures named in the ⟨*tagets*⟩ list as specified in the ⟨*tmp_hp*⟩ parameter, which is a `HitPointTmpHPRequest` object. You shoud not normally provide values for ⟨*requested_by*⟩ or

⟨*requesting_client*⟩ yourself. These will be provided by the server and will appear when the handler invokes the corresponding version of this method upon receipt. The ⟨*request_id*⟩ is an aribtrary ID you wish to use to track this request. The server should (but is not guaranteed to) respond with either a `failed` message (if your request was denied) or a `hit_point_acknowledge` message (if it was approved).

⟨*s*⟩`.load_from(`⟨*server_id*⟩`,` ⟨*cache_only*⟩`=False,` ⟨*merge*⟩`=False,` ⟨*is_local_file*⟩`=False)`

Given the ⟨*server_id*⟩ of a map file, this commands all clients to load that file into their map, merging that data with the map's current contents. This ensures that the client has a cached version of the file, downloading one if necessary. If the ⟨*cache_only*⟩ flag is true, then the fetching and caching operations are carried out, but the contents are not actually loaded. If ⟨*merge*⟩ is true, then the client should load the contents of the file without removing what's already there.

⟨*s*⟩`.marco()`

Send a ping message to all clients. Calling this method raises an exception because this is something only the server can do. However, it's included to preserve the symmetry between methods implemented here and in your handler. When the server pings you, your ⟨*handler*⟩`.marco()` method will be called. The appropriate response is for you to invoke the ⟨*s*⟩`.polo()` method.

⟨*s*⟩`.mark(`⟨*x*⟩`,` ⟨*y*⟩`)`

Place a temporary animated marker at coordinates (⟨*x*⟩, ⟨*y*⟩) on the map to draw attention to that location.

⟨*s*⟩`.place_someone(`⟨*obj_id*⟩`,` ⟨*color*⟩`,` ⟨*name*⟩`,` ⟨*area*⟩`,` ⟨*size*⟩`,` ⟨*obj_type*⟩`,` ⟨*gx*⟩`,` ⟨*gy*⟩`,` ⟨*reach*⟩`=F`

This introduces a new creature token to the maps. The ⟨*obj_id*⟩ parameter is the unique ID by which the creature will be known (a good practice is to generate a UUID and use the hex value (without other punctuation) as the ⟨*obj_id*⟩. The ⟨*gx*⟩ and ⟨*gy*⟩ are floating-point or integer values giving the position of the creature token in ⟨*numbers of squares*⟩ on the map. The ⟨*reach*⟩ flag indicates if they are using a reach weapon. The other parameters are self-explanatory, providing the creature's name, threat area (and color to shade in that area), and size category and more fully documented in `gma-mapper`(6). If a creature with the same ⟨*name*⟩ already exists, this will replace it.

⟨*s*⟩`.polo()`

Send an acknowledgement of the server's ping signal so it knows you're still alive.

⟨*s*⟩`.query_dice_presets(`⟨*for_user*⟩`=None,` ⟨*globals_only*⟩`=False)`

Requests that the server send your current die-roll preset list to you.

This should result in receipt of an `update_dice_presets` message. If ⟨*globals_only*⟩ is true, requests that only the global (system-wide) presets be sent. Otherwise, the target user's and the global set will be sent.

⟨*s*⟩`.query_image(`⟨*image_def*⟩`)`
Ask if the server or a peer knows of the definition of the image with the given definition (as an `ImageDefinition` object), at any of the the zoom factors listed in that object. If anyone does, you should receive (at some future point in time) an `add_image` response which your ⟨*handler*⟩`.add_image()` method can handle. There is no negative response (other than silence) since this query could be answered by any arbitrary client rather than a central server. If you receive this query yourself, and you know the definition of that image, you should respond by calling the ⟨*s*⟩`.add_image()` method to inform your peer of the information they requested.

⟨*s*⟩`.query_peers()`
Ask the server to send you a list of currently-connected peers. This should result in receipt of a `update_peer_list` message.

⟨*s*⟩`.remove_obj_attributes(`⟨*obj_id*⟩`,` ⟨*attrname*⟩`,` ⟨*values*⟩`)`
Just like ⟨*s*⟩`.add_obj_attributes()` but removes the ⟨*values*⟩ from the object's attribute if they are present.

⟨*s*⟩`.roll_dice(`⟨*roll_spec*⟩`,` ⟨*recipients*⟩`=None,` ⟨*to_all*⟩`=False,` ⟨*blind_to_gm*⟩`=False,` ⟨*request_id*⟩`=None)`

Roll the dice specified by ⟨*roll_spec*⟩ (see `gma-dice(3)` for details as to how this can be specified). If no recipients are listed at all, the results are sent back to you in the form of a `roll_result()` message, and to no one else. Otherwise, if ⟨*blind_to_gm*⟩ is `True`, then only the GM will receive the results of the roll, but you will be sent a `roll_result()` message with everything except the results of the roll. In this case, the ⟨*result_suppressed*⟩ parameter will be true to indicate that the result value should be ignored and that no actual results will appear in this messsage. Otherwise, if ⟨*to_all*⟩ is `True`, then the results will be sent to every logged-in peer. Otherwise, the results will be sent to the list of user names in the ⟨*recipients*⟩ list. The GM always has the username "GM".

If you sent a die-roll request which could not be processed by the server, you will receive a `roll_result()` message with the ⟨*invalid_request*⟩ paramter true. The result details will contain a message to the user to explain the issue. The flags ⟨*origin*⟩ and ⟨*replay*⟩ have the same meaning as they do for the `chat_message` server message.

⟨*s*⟩`.roll_result(`⟨*title*⟩`,` ⟨*result*⟩`,` ⟨*details*⟩`,` ⟨*sender*⟩`,` ⟨*recipients*⟩`=None,` ⟨*to_all*⟩`=False,` ⟨*blind_to_gm*⟩`=Fal`

Report the result from a previous call to `roll_dice()`. The ⟨*title*⟩ is assigned by the user who rolled the dice (but may be blank). The ⟨*result*⟩ is a numeric result of the die roll. The ⟨*details*⟩ parameter is a list of tuples describing how the ⟨*result*⟩ was arrived at. (See `gma-dice`(3) for details.) The ⟨*sender*⟩ parameter indicates the user name of the person who made the die roll. The other parameters have the same meaning as they do for the `roll_dice()` method. Calling this method raises an exception because this is something only the server can do. However, it's included to preserve the symmetry between methods implemented here and in your handler. The dispatcher will call your ⟨*handler*⟩`.roll_result()` method as needed.

⟨*s*⟩`.subscribe(`⟨*msg_list*⟩`=None)`
: Subscribe to the set of server messages listed in ⟨*msg_list*⟩. This arranges for the server to send only the subscribed messages, so the client doesn't need to receive them only to ignore them. If ⟨*msg_list*⟩ is a sequence type (such as a `set`), its contents are assumed to be a set of strings with the same values as the names of the message-handling methods documented here. The class attribute `all_messages` provides a set of all possible such strings. If ⟨*msg_list*⟩ is a `MapperProtocolHandler` object such as would be passed to the `dispatch()` method, all messages with defined handler methods in that object are subscribed. Thus, you can simply pass your dispatcher object to this method to automatically subscribe to all messages you intend to handle in your client application. The set of currently-subscribed messages may be read from the ⟨*s*⟩`.subscribed_messages` attribute. If ⟨*msg_list*⟩ is `None` or missing, the default behavior is to subscribe to all possible messages from the server.

⟨*s*⟩`.sync()`
: Request that the server send you all of the current game state. Doing this from a blank map will load all the current mapper contents onto it. The server may be configured to perform this sync operation automatically when you connect.

⟨*s*⟩`.sync_chat(`⟨*target*⟩`=None)`
: Request that the server send you its stored history of chat messages (`TO`, `ROLL`, and `CC` message types). If ⟨*target*⟩ is given as an integer value, then rather than sending all messages, only send the most recent *n* messages (if the value is a negative integer –*n*), or all messages with `ID`s greater than ⟨*target*⟩.

⟨*s*⟩`.toolbar(`⟨*enabled*⟩`)`
: Command all peers to turn on (if ⟨*enabled*⟩ is `True`) or off (otherwise) their toolbars.

⟨*s*⟩`.update_clock(`⟨*abs*⟩`,` ⟨*rel*⟩`,` ⟨*running*⟩`)`
: Inform clients that the game clock now reads ⟨*abs*⟩ seconds from the

GMA clock's epoch, and ⟨*rel*⟩ seconds from some arbitrary point of reference (often, the start of combat). Both values are floating-point. Calling this method raises an exception because this is something only the server can do. However, it's included to preserve the symmetry between methods implemented here and in your handler. When the clock is changed, the dispatcher will call your ⟨*handler*⟩`.update_clock()` method. If ⟨*running*⟩ is true, the local client should keep running the clock forward in real time after updating the time to the indicated values. Clients should not make these real time updates while in combat mode.

⟨*s*⟩`.update_dice_presets(`⟨*preset_list*⟩`,` ⟨*for_user*⟩`=None,` ⟨*delegates*⟩`=None,` ⟨*delegate_for*⟩`=None,` ⟨*globals_on*

The client should accept the die-roll presets defined in ⟨*preset_list*⟩ as the new set of presets in effect, replacing whatever else was previously defined. If the ⟨*preset_details*⟩ parameter to the `SocketInterface` constructor was `False`, this value will be a `DieRollPresetCollection` object, which is a list of `DieRollPreset` values, each of which are 3-string tuples describing die-roll presets, the exact interpretation of which can be somewhat client-dependent. If ⟨*preset_details*⟩ was `True`, then this will be a `DieRollObjectCollection` object instead, which is a list of `DieRollObject` object subclasses such as (for example, but not necessariy limited to) `SimplePreset`, `DictPreset`, `DieRollModifier`, or `LookupTable`, which are more detailed objects with their own attributes and methods which provide specific access to the details appropriate to that preset's information and behavior. See **gma-dice**(3) for details. Calling this method raises an exception because this is something only the server can do. However, it's included to preserve the symmetry between methods implemented here and in your handler. When the server's preset list is changed, or in response to you calling `query_dice_presets()`, the dispatcher will call your ⟨*handler*⟩`.update_dice_presets()` method. When this method is called via the handler, ⟨*globals_only*⟩ will be true if the presets in ⟨*global_presets*⟩ are the only items being sent. In this case, the server is only updating you with the system-wide presets and you should make no changes to the user's own preset list at all. Otherwise, you should consider ⟨*preset_list*⟩ to be the new complete set of the user's personal preset list and ⟨*global_presets*⟩ to be the new complete set of the global list.

⟨*s*⟩`.update_initiative_list(`⟨*slot_list*⟩`)`

Updates the initiative order listing for all combatants. Each element in ⟨*slot_list*⟩ describes the occupant of one initiative slot (unoccupied slots will not be described at all). Each of these is a `MapInitiativeSlot` object.

⟨*s*⟩`.update_obj_attributes(`⟨*obj_id*⟩`,` ⟨*kvlist*⟩`)`

For an existing map element or creature with the given ⟨*obj_id*⟩, up-

date its attributes from the dictionary ⟨*kvlist*⟩. This dictionary maps attribute names to the new values for those attributes. Attributes not in ⟨*kvlist*⟩ should remain as-is.

⟨*s*⟩`.update_peer_list(`⟨*peer_list*⟩`)`
Notify the client of the current list of peers connected to the server, in the ⟨*peer_list*⟩ parameter. Each element of this list is a `Peer` object.

Calling this method raises an exception because this is something only the server can do. However, it's included to preserve the symmetry between methods implemented here and in your handler. If the server sends you an updated peer list, the dispatcher will call your ⟨*handler*⟩`.update_peer_list()` method.

⟨*s*⟩`.update_status_marker(`⟨*condition*⟩`,` ⟨*shape*⟩`,` ⟨*color*⟩`,` ⟨*description*⟩`="")`

Add (or redefine) the client's way of marking that a creature has the named ⟨*condition*⟩. This may define a condition that had previously not been defined at all. The ⟨*shape*⟩ is one of the marker shape symbols defined in `gma-mapper`(6). The ⟨*color*⟩ is the color in which to draw the marker. Calling this method raises an exception because this is something only the server can do. However, it's included to preserve the symmetry between methods implemented here and in your handler. The server normally only sends these upon login, at which time the `SocketInterface` object stores them in its `conditions` dictionary, so ordinarily you should never even receive one of these messages. If for some reason you do, the dispatcher will call your ⟨*handler*⟩`.update_status_marker()` method.

⟨*s*⟩`.update_turn(`⟨*actor_id*⟩`,` ⟨*rounds*⟩`,` ⟨*hours*⟩`,` ⟨*minutes*⟩`,` ⟨*seconds*⟩`,` ⟨*count*⟩`)`

Declares whose turn it is in combat mode. ⟨*actor_id*⟩ is the name or ID of a creature or the special name "`*Monsters*`" to indicate all monsters, or may be the empty string to indicate that it is not anyone's turn at the moment. The other values indicate the current number of rounds, hours, minutes, and seconds that have elapsed since the start of combat, as well as the initiative count within the round (a number from 0 to 59).

Low-Level Interface

Once constructed, a `SocketInterface` object ⟨*s*⟩ provides the following methods which provide a direct access to sending and receiving data on the socket connection:

⟨*s*⟩`.sendall(`⟨*data*⟩`)`
Transmit ⟨*data*⟩ to the map server. The data string sent must conform to the protocol defined for the map service. If an error results

from trying to send the data, a new connection to the server will be attempted. A newline is not automatically added as ⟨*data*⟩ is transmitted; the caller must supply one.

⟨*s*⟩`.hold()`
: Calls to ⟨*s*⟩`.sendall()` will not be transmitted but will rather be held in a buffer until the ⟨*s*⟩`.release()` method is called.

⟨*s*⟩`.release()`
: Release held messages since the previous call to ⟨*s*⟩`.hold()` but it will discard (and not transmit) any `CO`, `CS`, `I`, or `IL` commands other than the most recent of each of those.

⟨*s*⟩`.recv(`⟨*size*⟩`)`
: Read up to ⟨*size*⟩ bytes from the server. This is a raw socket `recv()` call. For a higher-level interface, the ⟨*s*⟩`.read_poll()` method is recommended instead.

⟨*s*⟩`.reconnect()`
: Close the current connection to the server and try to establish a new one.

⟨*s*⟩`.read_poll(`⟨*verbose*⟩`=False, `⟨*timeout*⟩`=0)`
: If a complete line of data is available from the server, it is read and returned to the caller. Otherwise, `None` is returned. This method will read all it possibly can, but will only return complete lines as they are available every time it is called. The newline sent by the server at the end of the line is not stripped before the line is returned to the caller.

  If ⟨*timeout*⟩ is 0, this method will return immediately if there isn't a full command ready to be processed already. Otherwise, ⟨*timeout*⟩ specifies the number of seconds to wait for more incoming data before giving up, returning `None` if a full line wasn't received by that time. This value may be floating-point.

  As a special case, if ⟨*timeout*⟩ is `None`, then the method will wait indefinitely until some data are received from the server; however, if it receives an incomplete line from the server it will not continue to wait for more but will return `None` at that point. A subsequent call to `read_poll()` will collect the rest of the incomplete line and return it.

Other Data Structures

The `MapCondition` and `MapCharacter` objects are simple structures which simply contain data attributes matching the names as shown in the constructor. Thus, given a character in a `MapCharacter` object ⟨*c*⟩, one may examine ⟨*c*⟩`.name` to see their name, ⟨*c*⟩`.color` to see the color of their threat area, etc.

DIAGNOSTICS

A `SocketInterfaceError` exception may be thrown to indicate problems specifically with the interface, in addition to other standard exceptions.

EXAMPLE

The following simple example responds to server pings and prints server comments and who is currently logged in. It also prints chat messages and every minute sends a chat message of its own.

```
from MadScienceZone.GMA.SocketInterface \
   import SocketInterface, MapperprotocolHandler
from MadScienceZone.GMA.Mapper.Authentication \
   import Authenticator
import time

class MapEvents (MapperProtocolHandler):
  def marco(self):
    global server
    server.polo()

  def comment(self, text):
    print(f'server says: {text}')

  def chat_message(self, message, sender, recipients,
                   to_all, message_id):
    if to_all:
      print(f'{sender}: {message}')
    else:
      print(f'{sender} (to {recipients}): {message}')

  def update_peer_list(self, peer_list):
    print('WHOM ADDRESS------------- USER---- CLIENT---- AUTH')
    for p in peer_list:
      print(f'{"me" if p.is_me else "peer":4} {p.addr:20} \
{p.user:8} {p.client:10} {"yes" if p.is_auth else "no"}')

handler = MapEvents()
auth = Authenticator(secret=b'cookies', client='test program',
                     username='alice')
server = SocketInterface(('example.com', 2323),
                     authenticator=auth, subscribe_to=handler)

while (True):
```

```
    server.dispatch(handler)
    if int(time.time()) % 60 == 0:
        server.chat_message('All is well.', to_all=True)
    time.sleep(1)
```

SEE ALSO

`gma-authenticator`(3), `gma-mapobject`(3), `gma-mapper`(5), `gma-mapper`(6).

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

COMPATIBILITY NOTES

The 5.0 release introduced some breaking changes to the API in addition to the underlying protocol being very different.

In this version, some bare lists of values have been replaced to input or output objects such as `DieRollPreset`, `DieRollPresetCollection`, `ImageDefinition`, `ImageInstance`, `MapCondition`, `MapInitiativeSlot`, `PackageVersion`, and `Peer`.

Server messages `cache_file`, `load_file`, and `load_object` no longer exist, so the corresponding names and methods have been removed. The new messages `echo`, `load_arc`, `load_circle`, `load_line`, `load_polygon`, `load_rectangle`, `load_spell_area_of_effect`, `load_text`, `load_tile`, and `update_progress` have been added.

The following methods have been changed in the API for sending and receiving messages:

`add_character`
: has been removed.

`add_image`
: now takes an `ImageDefinition` object as its parameter.

`query_image`
: now takes an `ImageDefinition` object as its parameter.

`clear_from`
: added the ⟨*is_local_file*⟩ parameter.

`update_peer_list`
: now takes a list of `Peer` objects.

`update_dice_presets`
: now takes a list of `DieRollPreset` objects.

`update_status_marker`
: now includes a fourth parameter, ⟨*description*⟩.

`echo`
: was added.

`update_initiative_list`
: now takes a list of `MapInitiativeSlot` objects.

`place_someone`
: is still undergoing changes.

`update_progress`
: was added.

`toolbar`
: the ⟨*flag*⟩ parameter is now called ⟨*enabled*⟩.

`combat_mode`
: the ⟨*flag*⟩ parameter is now called ⟨*enabled*⟩.

`define_dice_presets`
: now takes a `DieRollPresetCollection` value or a list of `DieRollPreset` values.

`roll_dice`
: The use of `*`, `%`, and `@` in recipient lists is deprecated. Additional fields have been added.

`roll_result`
: Additional fields have been added.

The `Peer` object has changed (dropped the ⟨*is_main*⟩ and ⟨*is_write_only*⟩ fields).

Note that any functions which work with map elements (objects drawn on-screen) are still in flux and subject to change.

BUGS

The decision to have `SocketInterface` prompt the user for their password on behalf of the `Authenticator` object seems questionable. It is this way because the socket interface code is already interactive and has callback support for user input, while the authenticator object is more self-contained, but logically it probably should be the authenticator which does that password management.

gma-strip_ceature_base_name(3)

See gma-mapservice(3), p. 284.

gma-tcllist(3)

NAME

TclList – Python interface to Tcl list objects

SYNOPSIS

```
import MadScienceZone.Common.TclList

⟨tl⟩ = TclList(⟨tcl_string⟩=, ⟨make_numeric⟩=False,
⟨from_list⟩=None)
⟨tl⟩.make_numeric()
⟨tl⟩.from_list(⟨iterable⟩)
⟨string⟩ = ⟨tl⟩.to_tcl()
```

DESCRIPTION

Some of the older elements of GMA (which used to be entirely written in the Tcl language, after it was ported from the even older C++ code) use Tcl list objects as their data representation.

While this is obviously convenient for Tcl programs in that they can take such strings and natively use them as lists of values, it is also useful generally in that it is a simple string representation of a simple data structure. The actual definition of this string format is included below.

The `TclList` Python object provides an easy Python interface to manipulate Tcl lists as Python lists. (Courtesy of the fact that the Python's `tkinter` standard library module includes an embedded Tcl interpreter, this is done by calling into the Tcl interpreter, so this should be very accurate in terms of being genuine Tcl lists and not an approximate emulation.)

To create a `TclList` object from a string representation of a Tcl list (such as a value read from a map file), simply call the constructor

⟨*tl*⟩ = `TclList(`⟨*tcl_string*⟩`)`

Now you can access ⟨*tl*⟩ as if it were a Python `list` object. For example, after executing

```
tl = TclList('1 2 {Hello world} {3 4 {5 6}}')
```

then the contents of `tl` will be `['1', '2', 'Hello world', '3 4 {5 6}']`.

A couple of things to note here. First, since no type information is provided in the Tcl list representation (and in fact Tcl uses dynamic weak type bindings anyway), all values are assumed to be strings. Second, there is no way to distinguish a string which contains spaces from a sub-list, so "{Hello world}" may be a string of two words or a list with two elements. You are expected to simply use each appropriately in each case. If we wanted to

consider element 3 of `tl` in this example as a list, we can construct a new `TclList` for it:

```
tl2 = TclList(tl[3])
```

which gives us `tl2=['3', '4', '5 6']`.

If the constructor's optional ⟨*make_numeric*⟩ parameter is `True`, it will convert every element which appears to be a numeric value into a Python `int` or `float` object:

```
tl = TclList('1 2 {Hello world} {3 4 {5 6}}', make_numeric=True)
```

in this case, the contents of `tl` will be `[1, 2, 'Hello world', '3 4 {5 6}']`.

You can also invoke the `tl.make_numeric()` method later to convert numeric elements at that time.

Alternatively, if you have a Python iterable object such as a list or tuple, you can call the constructor as

⟨*tl*⟩ = `TclList(from_list=`⟨*iterable*⟩`)`

in which case, the `TclList` object will be created from the ⟨*iterable*⟩. If the ⟨*from_list*⟩ parameter is given, then ⟨*tcl_string*⟩ is ignored. The ⟨*make_numeric*⟩ parameter is still respected and will force elements to numeric where possible if it is `True`.

If you already have a `TclList` object ⟨*tl*⟩, you may call the ⟨*tl*⟩`.from_list()` method with an ⟨*iterable*⟩ parameter. This will replace the existing elements of ⟨*tl*⟩ with those from the iterable. For example,

```
tl=TclList()
tl.from_list([1, 2, [3, ['a', 'b'], 4, 5],
              'spam spam eggs spam'])
```

or

```
tl=TclList(from_list=[1, 2, [3, ['a', 'b'], 4, 5],
                      'spam spam eggs spam'])
```

both yield values in `tl` of

```
[1, 2, '3 {a b} 4 5', 'spam spam eggs spam']
```

A `TclList` object may be converted to a valid Tcl string representation by calling the ⟨*tl*⟩`.to_tcl()` method. For example, given the ⟨*tl*⟩ object from the previous example, calling ⟨*tlist*⟩ = ⟨*tl*⟩`.to_tcl()` will assign the string "`1 2 {3 {a b} 4 5} {spam spam eggs spam}`" to ⟨*tlist*⟩.

TCL LIST FORMAT

In a nutshell, a Tcl list (as a string representation) is a space-delimited list of values. Any value which includes spaces is enclosed in curly braces. An empty string (empty list) as an element in a list is represented as "`{}`". (E.g., "`1 {} 2`" is a list of three elements, the middle of which is an empty string.) An entirely empty Tcl list is represented as an empty string "".

A list value must have balanced braces. A balanced pair of braces that happen to be inside a larger string value may be left as-is, since a string that happens to contain spaces or braces is only distinguished from a deeply-nested list value when you attempt to interpret it as one or another in the code. Thus, the list

"`a b {this {is a} string}`"

has three elements: "`a`", "`b`", and "`this {is a} string`". Otherwise, a lone brace that's part of a string value should be escaped with a backslash:

"`a b {this \{ too}`"

Literal backslashes may be escaped with a backslash as well.

While extra spaces are ignored when parsing lists into elements, a properly formed string representation of a list will have the miminum number of spaces and braces needed to describe the list structure.

DIAGNOSTICS

These functions may raise the following exceptions in addition to the normal set that may occur otherwise:

`TclListFormatError` The string does not represent a valid Tcl list.

SEE ALSO

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

The Go port appeared in November, 2020, in GMA version 4.2.2. The original Python version calls directly into the Python interpreter's embedded Tcl interpreter to perform the required translations. The Go version implements the conversion operations directly, including a minimal amount of code ported from the open-source Tcl interpreter code.

BUGS

gma-to_roman(3)

See gma-romannumerals(3), p. 318.

gma-ustar(3)

NAME

ustar – Pure Tcl USTAR Archive Reader

SYNOPSIS

(If package installed globally)
`package require ustar`
(Otherwise)
`source ustar.tcl`

`::ustar::contents` ⟨*stream*⟩

`::ustar::file_contents` ⟨*path*⟩

`::ustar::gzip_contents` ⟨*path*⟩

`::ustar::extract` ⟨*stream*⟩ ⟨*callback*⟩

`::ustar::file_extract` ⟨*path*⟩ ⟨*callback*⟩

`::ustar::gzip_extract` ⟨*path*⟩ ⟨*callback*⟩

`::ustar::format_contents` ⟨*contents*⟩

DESCRIPTION

This module provides a simple interface for reading USTAR archive files, to either enumerate their contents or extract them completely. The following procedures are provided.

`::ustar::contents` ⟨*stream*⟩
: Scan the archive by reading data from ⟨*stream*⟩ (which must be a stream channel open in binary mode). The stream is not closed at the end of the operation. This returns a list of metadata values, one per file, in the order in which they appear in the archive. Each is a list of 13 values:

  ⟨***size***⟩ — The length, in bytes, of the file. Things without lengths (such as directories) will have a zero in this field.

  ⟨***type***⟩ — The file type. This is a single character exactly as found in the USTAR header field. A null byte or ASCII `0` digit indicates a regular file. Other than null, the

type characters will be ASCII alphanumeric characters. A `1` indicates the file is a hard link to another file already recorded in the archive. A `2` indicates that this entry describes a symbolic link. Types `3`, `4`, `5`, and `6` indicate character devices, block devices, directories, and FIFOs respectively. Other type values may be used to designate vendor-specific custom types of files.

⟨***name***⟩ The full pathname of the file. This might contain arbitrary Unicode characters.

⟨***mode***⟩ The binary file mode as described in `lstat`(2).

⟨***uid***⟩ The numeric user ID of the file's owner.

⟨***uname***⟩ If provided in the archive, the user name of the file's owner.

⟨***gid***⟩ The numeric group ID of the file's owner.

⟨***gname***⟩ If provided in the archive, the group name of the file's owner.

⟨***mtime***⟩ The file's modification time as a UNIX `time_t` value. If the archive supports it, this may be a floating-point value giving fractional seconds.

⟨***link***⟩ For links, this is the pathname of the file being linked to.

⟨***major***⟩ For devices, this is the numeric major device number. Otherwise zero.

⟨***minor***⟩ For devices, this is the numeric minor device number. Otherwise zero.

⟨***attrs***⟩ If any extended attributes were given for the file, they are listed here. This is a list with an even number of elements, which are taken in pairs as the name of the attribute followed by its value. These are completely arbitrary. Any which map directly to the above metadata fields are automatically updated in the other fields, but regardless of that, the entire extended attribute set appears here.

`::ustar::file_contents` ⟨*path*⟩
: As an convenience function, this opens ⟨*path*⟩ as a binary file and then calls `contents` on it, returning the result. This does close the file when finished reading from it.

`::ustar::gzip_contents` ⟨*path*⟩
: As with `file_contents` but decompresses the data as described for `gzip_extract` below.

`::ustar::extract` ⟨*stream*⟩ ⟨*callback*⟩
: This reads the archive data from the open binary file stream channel ⟨*stream*⟩ like the `contents` procedure does, but it also extracts the actual file data in addition to the metadata for each file. As each file is read, the supplied ⟨*callback*⟩ procedure is invoked to handle the disposition of the file's data, so the caller can decide whether to process the data in-memory or save it somewhere, perhaps after other processing takes place. The callback is invoked with two arguments: the metadata as already described above for the `contents` procedure, and the binary data of the file itself.

`::ustar::file_extract` ⟨*path*⟩ ⟨*callback*⟩
: As an convenience function, this opens ⟨*path*⟩ as a binary file and then calls `extract` on it, saving the caller from the trouble of opening and closing the archive.

`::ustar::gzip_extract` ⟨*path*⟩ ⟨*callback*⟩
: As with `file_extract` but also decompresses the contents of ⟨*path*⟩ while reading from it. This uses a `gzip`(1)-compatible decompresion suitable for tar archives created with `tar`'s `-z` option.

`::ustar::format_contents` ⟨*contents*⟩
: Given a list of metadata values exactly as returned by the `contents` procedure, this returns a multi-line string with that information formatted in a human-readable presentation. Not all extended attributes are included.

DIAGNOSTICS

An exception is thrown if the archive is unable to be processed for reasons such as unexpected end of file, unparseable header fields, checksum verification failures, etc.

SEE ALSO

`gzip`(1), `lstat`(2), `pax`(1), `tar`(1).

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

Initial version created 16 July 2020.

BUGS

Does not support base-256-encoded numeric values in headers, nor global extended header blocks. Neither of those is commonly found enough to prioritize implementation but that may change in the future.

gma-versionutils(3)

NAME

VersionUtils – functions for dealing with program versioning

SYNOPSIS

```
import MadScienceZone.Common.VersionUtils

version_compare(⟨a⟩, ⟨b⟩)
```

DESCRIPTION

This module provides a set of utility functions for working with version numbers and upgrades.

The `version_compare()` function takes two strings, each of which consists of integer version numbers separated by dots. The first such number in each string is the most significant (major) version number, the next is the next-most significant sub-version, and so forth. Thus, "1.2" comes before "1.10", "2.2.4", comes before "2.3", etc. This function returns 0 if $\langle a \rangle$ and $\langle b \rangle$ are equal or equivalent. Otherwise it returns a number less than 0 if $\langle a \rangle$ is a lesser version number than $\langle b \rangle$, or a number greater than 0 if $\langle a \rangle$ is a greater version number than $\langle b \rangle$.

SEE ALSO

`gma-gmautil`(3).

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

Initial version created 02 August 2020.

gma-weightsandmeasures(3)

NAME

WeightsAndMeasures – Conversion functions

SYNOPSIS

```
import MadScienceZone.Common.WeightsAndMeasures
⟨lbs⟩, ⟨ilbs⟩, ⟨oz⟩ = grams_to_pounds(⟨g⟩)

⟨g⟩ = pounds_to_grams(⟨lb⟩)

⟨g⟩ = ounces_to_grams(⟨oz⟩)

⟨s⟩ = str_g2lb(⟨g⟩)

⟨cm⟩ = metricize_length(⟨v⟩)

⟨cm⟩ = feet_to_centimeters(⟨ft⟩)

⟨cm⟩ = inches_to_centimeters(⟨in⟩)

⟨ft⟩, ⟨in⟩ = centimeters_to_feet(⟨cm⟩)

⟨g⟩ = metricize_weight(⟨wt⟩)

⟨s⟩ = str_cm2ft(⟨cm⟩)
```

DESCRIPTION

These functions perform basic unit conversions of weights and measures.

The `grams_to_pounds()` function takes a number of grams (which may be a `float` value), and returns a tuple of three values: the equivalent number of pounds as a `float` value, the number of pounds truncated to an integer value, and the number of ounces in addition to the integer number of pounds. Thus, either the real number of pounds may be used, or the combination of the last two tuple elements gives the number of whole pounds and ounces separately.

The `pounds_to_grams()` function accepts a number of grams (which may be a string, integer, or float value), and returns the number of grams it's equivalent to as an integer value. The `ounces_to_grams()` function works identically, but the input is a number of ounces instead of pounds.

Given a number of grams, the `str_g2lb()` function returns a string representation in pounds and ounces in the form "⟨*n*⟩ oz", "⟨*m*⟩#", or "⟨*m*⟩# ⟨*n*⟩ oz".

The `metricize_length()` function converts a string value formatted as "⟨*n*⟩'" or "⟨*n*⟩'⟨*m*⟩"" (representing ⟨*n*⟩ feet and ⟨*m*⟩ inches) to an integer number of centimeters. If the value passed into this function is not a string, it is assumed to have already been converted and is simply truncated to an integer and returned.

The `feet_to_centimeters()` and `inches_to_centimeters()` functions make the corresponding conversions. Note that the result of either is truncated to an integer value.

The `centimeters_to_feet()` function returns the number of feet as a tuple of integers: the whole number of feet and number of inches beyond that.

The `metricize_weight()` function works just like `metricize_length()` does, accepting strings in the form produced by `str_g2lb()`.

The `str_cm2ft()` function works just like `str_g2lb()` does, but emits the weight in grams passed into it as a string in the format "⟨*n*⟩' ⟨*m*⟩"".

SEE ALSO

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

gma-write_map_file(3)

See gma-mapobject(3), p. 274.

gma-xy_to_map_label(3)

See gma-mapobject(3), p. 274.

gmacolors(3)

NAME

gmacolors – Color support utilities

SYNOPSIS

(If package installed globally)
`package require gmacolors`
(Otherwise)
`source gmacolors.tcl`

`::gmacolors::rgb_name` ⟨*color*⟩ ⇒ ⟨*name*⟩

DESCRIPTION

This module provides a set of miscellaneous utility functions dealing with colors.

`::gmacolors::rgb_name` ⟨*color*⟩
: Given a ⟨*color*⟩ code in the format #⟨*rrggbb*⟩ this function returns the X11 color code name corresponding to that RGB color value. If none is found, the original ⟨*color*⟩ string is returned.

AUTHOR

Steve Willoughby / steve@madscience.zone.

gmafile(3)

NAME

gmafile – GMA file access functions

SYNOPSIS

(If package installed globally)
`package require gmafile`
(Otherwise)
`source gmafile.tcl`

`::gmafile::default_arr` ⟨*aname*⟩ ⟨*id*⟩ ?`-value` ⟨*v*⟩? ?`--`? ⟨*attr*⟩ ?⟨*attr*⟩...?

`::gmafile::load_dice_presets_from_file` ⟨*f*⟩ ⇒ ⟨*meta*⟩ ⟨*plist*⟩

`::gmafile::load_from_file` ⟨*f*⟩ ⇒ ⟨*meta*⟩ ⟨*objlist*⟩

`::gmafile::load_legacy_map_data` ⟨*vlist*⟩ ⟨*meta*⟩ ⇒ ⟨*meta*⟩ ⟨*objlist*⟩

`::gmafile::load_legacy_map_file` ⟨*f*⟩ ⟨*ver*⟩ ⟨*oldmeta*⟩ ⇒ ⟨*meta*⟩ ⟨*objlist*⟩

`::gmafile::require_arr` ⟨*aname*⟩ ⟨*id*⟩ ⟨*attr*⟩ ?`attr`...?

`::gmafile::load_legacy_preset_file` ⟨*f*⟩ ⟨*ver*⟩ ⟨*oldmeta*⟩ ⇒ ⟨*meta*⟩ ⟨*plist*⟩

`::gmafile::save_arrays_to_file` ⟨*f*⟩ ⟨*meta*⟩ ⟨*elements*⟩ ⟨*elementtypes*⟩ ⟨*creatures*⟩

`::gmafile::save_dice_presets_to_file` ⟨*f*⟩ ⟨*objlist*⟩

`::gmafile::save_to_file` ⟨*f*⟩ ⟨*objlist*⟩

`::gmafile::upgrade_elements` ⟨*filedata*⟩ ⇒ ⟨*dictlist*⟩

DESCRIPTION

This module provides the functionality for reading and writing data files used by GMA, including map files and saved die-roll presets. Reading of legacy formats is supported, but new files will only be saved in the current format.

`::gmafile::default_arr` ⟨*aname*⟩ ⟨*id*⟩ ?`-value` ⟨*v*?⟩ ?`--`? ⟨*attr*⟩ ⟨?*attr*...?⟩

⟨*This function is intended for reading legacy files and may disappear in the future.*⟩ Ensures that a number of required entries are defined in the array named as ⟨*aname*⟩. For each listed attribute ⟨*attr*⟩, the array must contain a value associated with the key ⟨*attr*⟩:⟨*id*.⟩ If no such key is found, it will be created with the default value ⟨*v*⟩ provided via the `-value` option (or the empty string if no `-value` option was given).

The `--` option may be used to explicitly terminate the list of options in case `-value` might be the name of an attribute.

`::gmafile::load_dice_presets_from_file` ⟨*f*⟩

Given an open file stream ⟨*f*⟩, this reads a set of die-roll presets from that file, returning them as a list of two elements:

⟨*meta*⟩

A dictionary with the following fields, any of which may be empty or zero if they were not found in the file:

⟨*Timestamp*⟩

The time the file claims to have been generated, as a UNIX timestamp value (seconds since the epoch).

⟨*DateTime*⟩

The time the file claims to have been generated, as a human-readable string in arbitrary format. No attempt is made to reconcile this and ⟨*Timestamp*⟩, they are simply presented as they were in the file.

⟨*Comment*⟩

A comment that was stored in the file.

⟨*FileVersion*⟩

The file format version of the data read.

⟨*plist*⟩

A Tcl list of die-roll presets. Each of these is a dictionary with the following fields:

⟨*Name*⟩

The name of the die-roll preset as shown to the user. This name may include a vertical bar (|). If so, the text up to and including the bar should be hidden from the user, but used for the purposes of sorting the list on-screen.

⟨*Description*⟩

The description of what the preset is for.

⟨*DieRollSpec*⟩
: The actual specification for the die roller `dice`(3).

`::gmafile::load_from_file` ⟨*f*⟩
: Given an open file stream ⟨*f*⟩, this reads map file data from it and returns them as a list of two elements:

⟨*meta*⟩
: A dictionary of values describing what was just read. This has the following fields, any of which may be empty:

⟨*Timestamp*⟩
: The time the file claims to have been generated, as a UNIX timestamp value (seconds since the epoch).

⟨*DateTime*⟩
: The time the file claims to have been generated, as a human-readable string in arbitrary format. No attempt is made to reconcile this and ⟨*Timestamp*⟩, they are simply presented as they were in the file.

⟨*Comment*⟩
: A comment that was stored in the file.

⟨*Location*⟩
: The name of the location which the file represents.

⟨*FileVersion*⟩
: The file format version of the data read.

⟨*objlist*⟩
: A Tcl list of the objects loaded from the map file. Each element of this list is itself a Tcl list with the following two elements:

⟨*type*⟩
: The type of object. This will be one of the types as documented in `mapper`(5), such as `ARC`, `POLY`, `IMG`, `MAP`, `CREATURE`, etc.

⟨*d*⟩
: A dictionary which describes the object of the given type. The specific fields for each type are documented in the protocol specification in `mapper`(6).

`::gmafile::load_legacy_map_data` ⟨*vlist*⟩ ⟨*meta*⟩
: ⟨*This function is provided to handle legacy-format map data and may disappear in the future.*⟩ ⟨*You should never need to call this function directly; it is automatically invoked as needed by*⟩ `::gmafile::load_from_file`. Reads partially-parsed lines from ⟨*vlist*⟩. Each element of this list is a 2-element list where the first element is one of:

`IMG`
: The second element is a dictionary describing an image advertised by the server.

`MAP`
: The second element is a dictionary describing a map file advertised by the server.

`RAW`
: The second element is a raw, unparsed line of legacy-format data loaded from the file.

The ⟨*meta*⟩ parameter is a dictionary of metadata already loaded with information about the input data.

Returns a list in the same format as `::gmaproto::load_from_file`.

`::gmafile::load_legacy_map_file` ⟨*f*⟩ ⟨*ver*⟩ ⟨*oldmeta*⟩
: ⟨*This function is provided to handle legacy-format map data and may disappear in the future.*⟩ ⟨*You should never need to call this function directly; it is automatically invoked as needed by*⟩ `::gmafile::load_from_file`. Reads legacy-format data from the open file stream ⟨*f*⟩, which is already known to be in file format version ⟨*ver*⟩. The ⟨*oldmeta*⟩ value is a Tcl list with the old-style metadata as documented in the version 17 revision of `mapper`(5).

Returns a list in the same format as `::gmaproto::load_from_file`.

`::gmafile::require_arr` ⟨*aname*⟩ ⟨*id*⟩ ⟨*attr*⟩ ⟨*?attr...?*⟩
: This works just like `::gmafile::default_arr` except that it does not set default values into the array named as ⟨*aname*⟩. It only checks that the named attributes already exist in the array, stored under keys called ⟨*attr*⟩:⟨*id.*⟩

`::gmafile::load_legacy_preset_file` ⟨*f*⟩ ⟨*ver*⟩ ⟨*oldmeta*⟩
: ⟨*This function is provided to handle legacy-format preset data and may disappear in the future.*⟩ ⟨*You should never need to call this function directly; it is automatically invoked as needed by*⟩ `::gmafile::load_dice_presets_from_fi`

In a manner analogous to `::gmafile::load_legacy_map_file`, this reads data from the open stream ⟨*f*⟩ which are in legacy file format version ⟨*ver*⟩ and have the old-format metadata ⟨*oldmeta*⟩.

Returns the same values as `::gmafile::load_dice_presets_from_file`.

`::gmafile::save_arrays_to_file` ⟨*f*⟩ ⟨*meta*⟩ ⟨*elements*⟩ ⟨*elementtypes*⟩ ⟨*creatures*⟩

Given an open writable file stream ⟨*f*⟩, this saves the map data stored in the following array variable names to a mapper data file:

⟨*elements*⟩
: This names an array variable containing all of the map elements other

than creatures which appear in the mapper client. The keys for each are the object's ID, which should also be the value stored in the object's ⟨*ID*⟩ field. The values in the array are the corresponding dictionary defining that object's attributes.

⟨*elementtypes*⟩
: This names an array variable containing a mapping of object IDs to their internal data type. There must be an element in this array for each element in the ⟨*elements*⟩ array. The type strings are the ones as known in the mapper code, not the server's protocol, as would be output by `::gmaproto::GMATypeToObjType`.

⟨*creatures*⟩
: This names an array variable containing a mapping of creature IDs to a dictionary describing each creature. The keys of this array are not referenced.

The ⟨*meta*⟩ parameter is a dictionary describing this collection of map data, and has the following fields:

⟨*Timestamp*⟩
: The UNIX timestamp representing the date and time when this map data were generated. If this field is missing from the dictionary, the current date and time will be used.

⟨*DateTime*⟩
: A human-readable string describing the ⟨*Timestamp*⟩ field's value. If this field is missing from the dictionary, one will automatically be generated based on the ⟨*Timestamp*⟩ value.

⟨*Comment*⟩
: Any comment you wish to be associated with this map file.

⟨*Location*⟩
: The name of the location which the file represents.

`::gmafile::save_dice_presets_to_file` ⟨*f*⟩ ⟨*objlist*⟩
: Given an open writable stream ⟨*f*⟩, this saves a collection of die-roll presets to disk. The ⟨*objlist*⟩ parameter is a Tcl list with two elements:

⟨*meta*⟩
: A dictionary describing this file, with the following fields:

⟨*Timestamp*⟩
: The UNIX timestamp representing the date and time when this preset data were generated. If this field is missing from the dictionary, the current date and time will be used.

⟨*DateTime*⟩
: A human-readable string describing the ⟨*Timestamp*⟩ field's value. If

this field is missing from the dictionary, one will automatically be generated based on the ⟨*Timestamp*⟩ value.

⟨*Comment*⟩
: Any comment you wish to be associated with this file.

⟨*plist*⟩
: A Tcl list of dictionaries, each describing a single die-roll preset with the same fields as documented for `::gmafile::load_dice_presets_from_file`.

`::gmafile::save_to_file` ⟨*f*⟩ ⟨*objlist*⟩
: As an alternative to `::gmafile::save_arrays_to_file`, this function writes the map data as found in ⟨*objlist*⟩, which is a Tcl list containing the following two elements:

⟨*meta*⟩
: A metadata dictionary as described for `::gmafile::save_arrays_to_file`.

⟨*elements*⟩
: A Tcl list of all map elements to be saved. Each element of this list is itself a Tcl list of two elements:

⟨*type*⟩
: The element type name as known in the mapper software.

⟨*d*⟩
: The dictionary describing the object.

`::gmafile::upgrade_elements` ⟨*filedata*⟩
: ⟨*This function is intended for reading legacy files and may disappear in the future.*⟩ Each element in ⟨*filedata*⟩ is a tuple as described for `::gmafile::load_legacy_map_data`. This function converts all entries of `RAW` type into their new-format equivalents. A new copy of ⟨*filedata*⟩ with those substitutions in place is returned.

DIAGNOSTICS

An exception is thrown if a serious error is encountered.

Messages are printed to standard output to indicate progress or provide debugging information.

SEE ALSO

`openssl`(1).

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

This document describes version 1.0 of the `gmafile` package, released in December 2022.

gmaproto(3)

NAME

gmaproto – GMA client/server communication functions

SYNOPSIS

(If package installed globally)
`package require gmaproto`
(Otherwise)
`source gmaproto.tcl`

`::gmaproto::add_image` ⟨*name*⟩ ⟨*sizes*⟩

`::gmaproto::add_obj_attributes` ⟨*id*⟩ ⟨*attr*⟩ ⟨*values*⟩

`::gmaproto::adjust_view` ⟨*x*⟩ ⟨*y*⟩

`::gmaproto::allow` ⟨*features*⟩

`::gmaproto::auth_response` ⟨*challenge*⟩ ⇒ ⟨*response*⟩

`::gmaproto::background_redial` ⟨*tries*⟩

`::gmaproto::chat_message` ⟨*message*⟩ ⟨*sender*⟩ ⟨*recipients*⟩ ⟨*to_all*⟩ ⟨*to_gm*⟩

`::gmaproto::clear` ⟨*id*⟩

`::gmaproto::clear_chat` ⟨*silent*⟩ ⟨*target*⟩

`::gmaproto::clear_from` ⟨*server_id*⟩

`::gmaproto::combat_mode` ⟨*bool*⟩

`::gmaproto::comment` ⟨*text*⟩

`::gmaproto::DEBUG` ⟨*msg*⟩

`::gmaproto::define_dice_presets` ⟨*plist*⟩ ⟨*append*⟩

`::gmaproto::dial` ⟨*host*⟩ ⟨*port*⟩ ⟨*user*⟩ ⟨*pass*⟩ ⟨*proxy*⟩ ⟨*pport*⟩ ⟨*puser*⟩ ⟨*ppass*⟩ ⟨*client*⟩

```
::gmaproto::filter_dice_presets ⟨regex⟩

::gmaproto::from_enum ⟨key⟩ ⟨int⟩ ⇒ ⟨value⟩

::gmaproto::GMATypeToObjType ⟨gtype⟩ ⇒ ⟨otype⟩

::gmaproto::int_bool ⟨bool⟩ ⇒ ⟨int⟩

::gmaproto::is_connected ⇒ ⟨bool⟩

::gmaproto::is_ready ⇒ ⟨bool⟩

::gmaproto::json_bool ⟨bool⟩ ⇒ ⟨str⟩

::gmaproto::ls ⟨type⟩ ⟨d⟩

::gmaproto::load_from ⟨server_id⟩ ⟨cache_only⟩ ⟨merge⟩

::gmaproto::mark ⟨x⟩ ⟨y⟩

::gmaproto::new_dict ⟨command⟩ ?⟨key⟩ ⟨value⟩? ?⟨key⟩
⟨value⟩...? ⇒ ⟨d⟩

::gmaproto::new_id ⇒ ⟨uuid⟩

::gmaproto::ObjTypeToGMAType ⟨otype⟩ ?-protocol? ⇒ ⟨gtype⟩

::gmaproto::place_someone_d ⟨d⟩

::gmaproto::polo

::gmaproto::query_dice_presets

::gmaproto::query_image ⟨name⟩ ⟨size⟩

::gmaproto::remove_obj_attributes ⟨id⟩ ⟨attr⟩ ⟨values⟩

::gmaproto::redial
```

```
::gmaproto::roll_dice ⟨spec⟩ ⟨recipients⟩ ⟨to_all⟩ ⟨blind_to_gm⟩

::gmaproto::set_debug cmd

::gmaproto::subscribe ⟨msglist⟩

::gmaproto::sync

::gmaproto::sync_chat ⟨target⟩

::gmaproto::to_enum ⟨key⟩ ⟨value⟩ ⇒ ⟨int⟩

::gmaproto::toolbar ⟨bool⟩

::gmaproto::update_clock ⟨abs⟩ ⟨rel⟩

::gmaproto::update_obj_attributes ⟨id⟩ ⟨kvdict⟩

::gmaproto::update_progress ⟨id⟩ ⟨title⟩ ⟨value⟩ ⟨max⟩ ⟨done⟩ ⇒
⟨id⟩

::gmaproto::update_status_marker ⟨condition⟩ ⟨shape⟩ ⟨color⟩
⟨description⟩

::gmaproto::write_only ⟨main⟩
```

DESCRIPTION

This module provides the functionality for communicating with the server, including low-level communications, legacy protocol translation, and dispatching incoming server commands to the mapper for execution.

`::gmaproto::add_image` ⟨*name*⟩ ⟨*sizes*⟩
: Send image information to the server for the image with the given ⟨*name*⟩ (the messsage ID as referenced in map files and in the client). The ⟨*sizes*⟩ parameter is a Tcl list of dictionaries describing each instance of each zoomed size of that image as documented in `mapper`(6). This sends the `AI` protocol command.

`::gmaproto::add_obj_attributes` ⟨*id*⟩ ⟨*attr*⟩ ⟨*values*⟩
: Tell the other clients to adjust the object with the given ⟨*id*⟩ by adding each string in the Tcl list ⟨*values*⟩ to their field named ⟨*attr*⟩, which must be a field that accepts a list of strings. This sends the `OA+` protocol command.

`::gmaproto::adjust_view` ⟨*x*⟩ ⟨*y*⟩
: Tell other clients to ajust their scrollbars so that the distance scrolled is ⟨*x*⟩ to the right and ⟨*y*⟩ down, where ⟨*x*⟩ and ⟨*y*⟩ are values between 0 and 1, which indicate the proportion of full range to move the scrollbars. This sends the `AV` protocol command.

`::gmaproto::allow` ⟨*features*⟩
: Tell the server that you will allow the use of the listed optional ⟨*features*⟩ (a Tcl list of feature name strings). This sends the `ALLOW` protocol command.

`::gmaproto::auth_response` ⟨*challenge*⟩
: ⟨*You should not normally need to invoke this procedure directly.*⟩ Given a binary ⟨*challenge*⟩ value from the server, this calculates and returns the binary response needed to authenticate to the server, using the password previously given to `::gmaproto::dial`. If the username given is still empty, it also attempts to find the local username and sets the configured username to that value.

`::gmaproto::background_redial` ⟨*tries*⟩
: ⟨*You should not normally need to invoke this procedure directly.*⟩ Attempts to reconnect to the server. If this fails, it schedules itself to run again in a few seconds, with the ⟨*tries*⟩ value incremented.

`::gmaproto::chat_message` ⟨*message*⟩ ⟨*sender*⟩ ⟨*recipients*⟩ ⟨*to_all*⟩ ⟨*to_gm*⟩
: Sends a chat ⟨*message*⟩ to other clients. Clients should not set the ⟨*sender*⟩ value. If ⟨*to_gm*⟩ is true, the message will be sent only to the `GM`; otherwise, if ⟨*to_all*⟩ is true, the message is sent to all clients; otherwise it is sent to the list of usernames in ⟨*recipients*⟩. This sends the `To` protocol command.

`::gmaproto::clear` ⟨*id*⟩
: Tells other clients to remove the object identified by ⟨*id*⟩ as documented in `mapper`(6). This sends the `CLR` protocol command.

`::gmaproto::clear_chat` ⟨*silent*⟩ ⟨*target*⟩
: Tells others to clear chat messages specified by the ⟨*target*⟩ value. If ⟨*silent*⟩ is true, ask them not to advertise that this was done. This sends the `CC` protocol command.

`::gmaproto::clear_from` ⟨*server_id*⟩
: Instruct other clients to remove all elements from the map file ⟨*server_id*⟩. This sends the `CLR@` protocol command.

`::gmaproto::combat_mode` ⟨*bool*⟩
: Tell others to set combat mode if ⟨*bool*⟩ is true, otherwise unset it. This sends the `CO` protocol command.

`::gmaproto::comment` ⟨*text*⟩
: Send ⟨*text*⟩ as a comment to the server, which is probably a pointless thing to do. This sends the `//` protocol command.

`::gmaproto::DEBUG` ⟨*msg*⟩
: Send ⟨*msg*⟩ to the callback function configured via `::gmaproto::set_debug`.

`::gmaproto::define_dice_presets` ⟨*plist*⟩ ⟨*append*⟩
: Send a new set of die-roll presets to the server for storage. The ⟨*plist*⟩ parameter is a Tcl list of dictionaries describing each preset as per `mapper`(6). If ⟨*append*⟩ is true, the elements in ⟨*plist*⟩ are added to the ones already on the server; otherwise they replace the server's current set. This sends the `DD` or `DD+` protocol command.

`::gmaproto::dial` ⟨*host*⟩ ⟨*port*⟩ ⟨*user*⟩ ⟨*password*⟩ ⟨*proxy*⟩ ⟨*pport*⟩ ⟨*puser*⟩ ⟨*ppass*⟩ ⟨*client*⟩

: This is the initial command you should call to establish a connection to the server on ⟨*host*⟩ at the TCP ⟨*port*⟩ specified. Once the connection is established, the client will authenticate as the given ⟨*user*⟩ and ⟨*password*⟩ and will note that the connecting client is called ⟨*client*⟩. If a SOCKS proxy is needed, ⟨*proxy*⟩, ⟨*pport*⟩, ⟨*puser*⟩, and ⟨*ppass*⟩ give the proxy host, port, and login credentials to use.

  If the connection is lost, this package will automatically try to reconnect using the same parameters.

  As incoming commands are received from the server, they are dispatched back to the application by calling a procedure named

  `::DoCommand`⟨*cmd*⟩ ⟨*d*⟩

  where ⟨*cmd*⟩ is the server's command name. The single parameter ⟨*d*⟩ is a dictionary holding the command's parameter set. For example, if the server sent an `AV` command, then `::DoCommandAV` would be called in the application.

  If that failed, either because the command does not exist in the application or because it threw an error, then an error-handling function is called:

  `::DoCommandError` ⟨*cmd*⟩ ⟨*d*⟩ ⟨*err*⟩

  where ⟨*cmd*⟩ is the original command name, ⟨*d*⟩ is the parameter dictionary, and ⟨*err*⟩ is the error message received.

`::gmaproto::filter_dice_presets` ⟨*regex*⟩
: Asks the server to remove all stored die-roll presets whose names match the regular expression ⟨*regex*⟩. This sends the `DD/` protocol command.

`::gmaproto::from_enum` ⟨*key*⟩ ⟨*int*⟩
: Converts the integer value ⟨*int*⟩ into the enum string corresponding to that value for the enumerated type ⟨*key*⟩ (`Dash`, `Join`, `MoveMode`, etc.).

`::gmaproto::GMATypeToObjType` ⟨*gtype*⟩
: Converts the object type name as used by the server to the corresponding name used inside the mapper application, and returns it.

`::gmaproto::int_bool` ⟨*bool*⟩
: Returns 1 if ⟨*bool*⟩ is true, otherwise returns 0.

`::gmaproto::is_connected`
: Returns true if the client has an active network connection to the server.

`::gmaproto::is_ready`
: Returns true if the client has an active network connection and has successfully completed the initial negotion and authentication with the server.

`::gmaproto::json_bool` ⟨*bool*⟩
: Returns the boolean value passed as the string `true` or `false`, suitable for JSON strings.

`::gmaproto::ls` ⟨*type*⟩ ⟨*d*⟩
: Sends a map object to peer clients. ⟨*Type*⟩ is the GMA protocol type name (`ARC`, `LINE`, etc.) and ⟨*d*⟩ is an appropriate dictionary value for that type. This sends the `LS-ARC`, `LS-LINE`, etc. protocol commands.

`::gmaproto::load_from` ⟨*server_id*⟩ ⟨*cache_only*⟩ ⟨*merge*⟩
: Tells other clients to load elements from the given server map file called ⟨*server_id*⟩. If ⟨*merge*⟩ is true, the contents of that file should be merged with the existing map contents instead of replacing them. If ⟨*cache_only*⟩ is true, tell the clients to cache a copy of the file without actually loading anything from it. This sends the `L` protocol command.

`::gmaproto::mark` ⟨*x*⟩ ⟨*y*⟩
: Tell other clients to visually mark the location with the given map coordinates. This sends the `MARK` protocol command.

`::gmaproto::new_dict` ⟨*command*⟩ ⟨?*key*⟩ ⟨*value*?⟩ ⟨?*key*⟩ ⟨*value*...?⟩

: Construct a new dictionary suitable to hold the parameters for the given ⟨*command*⟩, with all fields defaulted. Additionally, any ⟨*key*⟩ and ⟨*value*⟩ pairs specified set the given fields in the new dictionary. The dictionary value is returned.

`::gmaproto::new_id`
: Generate a new unique ID suitable for use as object identifiers, and returns it.

`::gmaproto::ObjTypeToGMAType` ⟨*otype*⟩ ?`-protocol`?
: Computes and returns the server type name corresponding to the

mapper internal type name ⟨*otype*⟩. If the `-protocol` option is given, the protocol command name (with `LS-` prefix) is returned instead of the base type name.

`::gmaproto::place_someone_d` ⟨*d*⟩
: Tells the other clients to place a creature token on the map as described. If another creature is already present with the same name, it is replaced by the new one. This sends the `PS` protocol command. The parameter is a dictionary holding a creature object.

`::gmaproto::polo`
: Sends a "still alive" response to the server, typically in response to receiving a `MARCO` command. This sends the `POLO` server command.

`::gmaproto::query_dice_presets`
: Asks the server to send all of the stored presets. This sends the `DR` protocol command.

`::gmaproto::query_image` ⟨*name*⟩ ⟨*size*⟩
: Asks the server and other clients if any of them have heard of the given image ⟨*name*⟩ at the requested zoom ⟨*size*⟩. This sends the `AI?` protocol command.

`::gmaproto::query_peers`
: Requests that the server send the list of currently-connected clients. This sends the `/CONN` protocol command.

`::gmaproto::redial`
: Try to reconnect using the parameters given with the initial `::gmaproto::dial` command. ⟨*You don't normally need to call this directly.*⟩

`::gmaproto::roll_dice` ⟨*spec*⟩ ⟨*recipients*⟩ ⟨*to_all*⟩ ⟨*blind_to_gm*⟩
: Ask the server to roll the dice indicated by ⟨*spec*⟩. If ⟨*blind_to_gm*⟩ is true, the results are visible only to the `GM`; otherwise, if ⟨*to_all*⟩ is true, the results are sent to everyone; otherwise only to the list of user names in ⟨*recipients*⟩. This sends the `D` protocol command.

`::gmaproto::set_debug` ⟨*cmd*⟩
: Protocol debugging statements will be sent by calling ⟨*cmd*⟩ with a string parameter.

`::gmaproto::subscribe` ⟨*msglist*⟩
: Tell the server that we're only interested in receiving the command names listed in ⟨*msglist*⟩. If ⟨*msglist*⟩ is empty, then tell the server we accept all messages. This sends the `ACCEPT` protocol command.

`::gmaproto::sync`
: Request that the server send a set of commands which will bring the client up to date with the current game state. This sends the `SYNC` protocol command.

`::gmaproto::sync_chat` ⟨*target*⟩
: Request that the server send some or all (per the ⟨*target*⟩) value of the historical chat messages to the client. This sends the `SYNC-CHAT` protocol command.

`::gmaproto::to_enum` ⟨*key*⟩ ⟨*value*⟩
: Converts and returns the integer associated with the ⟨*value*⟩ in the enumerated type ⟨*key*⟩.

`::gmaproto::toolbar` ⟨*bool*⟩
: Tells other clients to turn on their toolbars if ⟨*bool*⟩ is true; otherwise turn them off. This sends the `TB` protocol command.

`::gmaproto::update_clock` ⟨*absolute*⟩ ⟨*relative*⟩
: Updates the game clock to the given ⟨*absolute*⟩ and ⟨*relative*⟩ time values. This sends the `CS` protocol command.

`::gmaproto::update_obj_attributes` ⟨*id*⟩ ⟨*kvdict*⟩
: Instructs the other clients to update the state of the object with the given ⟨*id*⟩ by setting each of the objects fields named as keys in ⟨*kvdict*⟩ with their corresponding values. This sends the `OA` protocol command.

`::gmaproto::update_progress` ⟨*id*⟩ ⟨*title*⟩ ⟨*value*⟩ ⟨*max*⟩ ⟨*done*⟩
: Instructs peers to display a progress meter with the given ⟨*id*⟩ (creating a new one if that is not an existing progress bar ⟨*id*⟩). If ⟨*done*⟩ is true, this is notice that the progress bar is no longer needed.

  If ⟨*id*⟩ is specified as `*`, then a new `ID` will be generated.

  If ⟨*max*⟩ is zero or `*`, then we are saying we don't know what the maximum value will be and the client should give a progress bar that shows activity but not specific progress toward a known goal.

  This sends the `PROGRESS` potocol command.

  The progress indicator's ⟨*id*⟩ is returned.

`::gmaproto::update_status_marker` ⟨*condition*⟩ ⟨*shape*⟩ ⟨*color*⟩ ⟨*description*⟩

: Tells the server to define a new status marker for ⟨*condition*⟩ with the given ⟨*shape*⟩, ⟨*color*⟩, and ⟨*description*⟩. This sends the `DSM` protocol command.

`::gmaproto::write_only` ⟨*main*⟩
: Tells the server that this client no longer wishes to receive any messages from it. If ⟨*main*⟩ is true, the client is also signalling that it wishes to be the primary client in the conversation.

EXTERNAL HOOKS

This package will invoke the following procedures in the main application to carry out its operations, typically in response to having received a server command for the client to do something.

`::DEBUG` ⟨*level*⟩ ⟨*message*⟩
: This is called to report a diagnostic condition or provide some level of verbose detail about the operation of the communications package, except for the debugging of the actual protocol interactions, which is handled by the callback registered via `::gmaproto::set_debug`.

`::report_progress` ⟨*message*⟩
: Reports user-friendly progress information.

`::say` ⟨*message*⟩
: Displays an urgent message, probably in a modal dialog or alert box.

`::DoCommand`⟨*cmd*⟩ ⟨*params*⟩
: Handles the receipt of the server command ⟨*cmd*⟩ by the client. The ⟨*params*⟩ parameter is a dictionary of values as sent by the server.

`::DoCommandError` ⟨*cmd*⟩ ⟨*params*⟩ ⟨*err*⟩
: Handles any error encountered when trying to execute a `::DoCommand`⟨*cmd*⟩ call.

DIAGNOSTICS

An exception is thrown if a serious error is encountered.

Messages are printed to standard output to indicate progress or provide debugging information.

The registered debugging hook is also used to print debugging and diagnostic messages.

SEE ALSO

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

This document describes version 1.0 of the `gmaproto` package, released in December 2022.

gmautil(3)

NAME

gmautil – General Tcl Utility Routines

SYNOPSIS

(If package installed globally)
`package require gmautil`
(Otherwise)
`source gmautil.tcl`

`::gmautil::dassign` ⟨*dict*⟩ ⟨*key1*⟩ ⟨*var1*⟩ ?⟨*key2*⟩ ⟨*var2*⟩...?

`::gmautil::is_git` ⟨*path*⟩ ⇒ `bool`

`::gmautil::lpop` ⟨*var*⟩ ⟨*index*⟩ ⇒ ⟨*element*⟩

`::gmautil::my_arch` ⇒ ⟨*arch*⟩

`::gmautil::my_os` ⇒ ⟨*os*⟩

`::gmautil::rdist` ⟨*min*⟩ ⟨*max*⟩ ⟨*cmd*⟩ ⟨*arglist*⟩ ⟨*var*⟩ ?⟨*var*⟩...?

`::gmautil::upgrade` ⟨*destination*⟩ ⟨*tmp*⟩ ⟨*url*⟩ ⟨*file*⟩ ⟨*old*⟩ ⟨*new*⟩ ⟨*strip*⟩ ⟨*launch*⟩ ⟨*msg*⟩ ⟨*proxy*⟩ ⟨*curl*⟩

`::gmautil::verify` ⟨*data*⟩ ⟨*signature*⟩ ⇒ ⟨*bool*⟩

`::gmautil::version_compare` ⟨*v1*⟩ ⟨*v2*⟩ ⇒ cmp

DESCRIPTION

This module provides a set of miscellaneous utility functions that could be useful to more than one part of the overall product suite.

`::gmautil::dassign` ⟨*dict*⟩ ⟨*key1*⟩ ⟨*var1*⟩ ⟨?*key2*⟩ ⟨*var2...*?⟩

This is analogous to the Tcl built-in `lassign` command, but it operates on a dictionary value ⟨*dict*⟩ instead of a list. For each pair of arguments after the ⟨*dict*⟩ value, the named ⟨*key*⟩ is looked up in the dictionary and its value is assigned to the variable ⟨*var*⟩.

If any ⟨*key*⟩ is a space-separated list of names, they are considered to be a list of nested sub-keys to be fetched from the dictionary in the same manner as `dict get`. For example, given the ⟨*key*⟩ `{foo bar}`

it will retrieve the value stored under the key `foo`, which is itself a dictionary value, in which the key `bar` is looked up, and its value assigned to the corresponding ⟨*var*⟩ variable.

`::gmautil::is_git` ⟨*path*⟩
Returns true if the directory ⟨*path*⟩ resides inside a Git working directory tree.

`::gmautil::lpop` ⟨*var*⟩ ⟨*index*⟩
This removes the ⟨*index*⟩th element from the list variable ⟨*var*⟩, returning the removed element. This is intended to provide the basic functionality of the Tcl 8.7 built-in command `lpop`, and in fact if the Tcl interpreter is at least version 8.7 it simply calls the `lpop` provided by the language, which has additional capabilities not implemented by this version.

`::gmautil::my_arch`
Returns the name of the hardware architecture running the program, with naming convention standard to the GMA suite. Currently, it translates the Tcl runtime's machine names as obtained from `tcl_platform(machine)` with the following translation:

Tcl Name	GMA Name
`x86_64`	`amd64`

Any name not otherwise listed is returned as-is from the Tcl runtime. More names will likely be added to this translation list in the future.

`::gmautil::my_os`
Returns the name of the operating system running the program, with naming convention standard to the GMA suite. Currently, it translates the Tcl runtime's os names as obtained from `tcl_platform(os)` with the following translation:

Tcl Name	GMA Name
`Darwin`	`darwin`
`Linux`	`linux`
`FreeBSD`	`freebsd`

Any name not otherwise listed is returned as-is from the Tcl runtime. More names will likely be added to this translation list in the future.

`::gmautil::rdist` ⟨*min*⟩ ⟨*max*⟩ ⟨*cmd*⟩ ⟨*arglist*⟩ ⟨*var*⟩ ⟨?*var*...?⟩
This is a specialized version of `lassign`, which is intended to receive a number of parameters in ⟨*arglist*⟩ to a GMA command called ⟨*cmd*⟩.

It requires that ⟨*arglist*⟩ contain at least ⟨*min*⟩ ant at most ⟨*max*⟩ elements, generating an error if that condition is not met. The values in ⟨*arglist*⟩ are then distributed out into the list of named variables in the order given. If there are more ⟨*var*⟩ variable names than elements in ⟨*arglist*⟩, the remaining variables are set to the empty string. Note that ⟨*cmd*⟩ is only used for forming error messages.

`::gmautil::upgrade` ⟨*destination*⟩ ⟨*tmp*⟩ ⟨*url*⟩ ⟨*file*⟩ ⟨*old*⟩ ⟨*new*⟩ ⟨*strip*⟩ ⟨*launch*⟩ ⟨*msg*⟩ ⟨*proxy*⟩ ⟨*curl*⟩

This function facilitates automatic upgrades within an application. The application is downloaded using the CURL program whose pathname is given in ⟨*curl*⟩ (using a proxy server as specified by ⟨*proxy*⟩ if that value is non-empty). The file itself is obtained from the server directory indicated by the ⟨*url*⟩ value, with the base filename ⟨*file*⟩ and suffix `.tar.gz` and `.tar.gz.sig` (the latter being a cryptographic signature verifying the authenticity of the compressed tar file). These files are downloaded to the temporary directory ⟨*tmp*⟩.

Once the files are downloaded, the signature is checked, and then the files are extracted into the destination diretory ⟨*destination*⟩, although a leading ⟨*strip*⟩ (possibly prefixed with `./`) is removed from the names as they exist in the tar file.

The tar file is expected to contain a manifest file in its top-level directory, called `__checksums__`. Each line of this file consists of a SHA256 checksum in hex, whitespace, a type character (space for text files, `*` for binary files, `?` for portable-newline files, or `^` for BITS files), then the path of the file. Only binary files are supported at this time. Every file extracted from the archive must have an entry in this manifest file, and the installed file's checksum must match the manifest entry.

Once that is all accomplished, the downloaded files are removed and, if ⟨*launch*⟩ is non-empty, the program will attempt to start the program ⟨*launch*⟩ relative to ⟨*destination*⟩ and then the calling program is terminated.

⟨*old*⟩ and ⟨*new*⟩ are the current and to-be-installed version numbers. Warnings will be issued before downgrading. The routine will refuse to continue if the versions are equal to each other.

`::gmautil::verify` ⟨*data*⟩ ⟨*signature*⟩

Given a binary data string ⟨*data*⟩, return true if the binary cryptographic signature in ⟨*signature*⟩ is valid for that data. This uses our built-in product public key.

`::gmautil::version_compare` ⟨*v1*⟩ ⟨*v2*⟩

Compares two version number strings to see which order they should go in, chronologically. Each string must consist of one or more integers (as ASCII digit sequences) separated by dots. The first such number is the most significant (major) version, the next is the next-most

significant sub-version within the major one, and so forth. Thus, 1.2 comes before 1.10, 2.2.4 comes before 2.3, etc. Returns 0 if ⟨*v1*⟩ and ⟨*v2*⟩ are equal or equivalent, a number <0 if ⟨*v1*⟩<⟨*v2*⟩ (i.e., ⟨*v1*⟩ comes before ⟨*v2*⟩), or a number >0 if ⟨*v1*⟩>⟨*v2*⟩.

DIAGNOSTICS

An exception is thrown if a serious error is encountered.

SEE ALSO

`openssl`(1).

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

This document describes version 1.1 of the `gmautil` package, released in December 2022.

Initial version created 17 July 2020.

APPENDIX C

File Format Reference

> Words are, in my not-so-humble opinion, our most inexhaustible source of magic, capable of both inflicting injury and remedying it.
>
> —Albus Dumbledore

The following pages document the formats of the various files used by GMA. These manual entries are also available on the system where GMA is installed in the form of online manual pages ("manpages"), which can be viewed using the `man` command. These pages are in section 5, so following the conventions for Unix manpages, their names have "(5)" appended to them.

gma-character-record(5)

NAME

gma-character-record – data file format for character record sheets

SYNOPSIS

This document describes the file format GMA uses to store character record information. These files typically have a "`.gcr`" filename suffix.

DESCRIPTION

Note: The software which manages this information is undergoing active development and is yet incomplete. As a consequence of this, the information contained here is subject to change as the software design solidifies. Also, there are some elements of this file format which are temporary placeholders to compensate for yet-to-be-written features of the program.

The tracking of character record data is being migrated more and more into the database, but much of it still remains stored in ASCII files as the database development continues. (That, by the way, is a tragically optimistic bit of fantasy. We're still using ASCII files.)

This document describes the ASCII file format that is currently in use. As the development of GMA continues, this file will be used more as a data interchange method instead of the primary storage for character data.

FILE FORMAT

The overall format is that of a classic "INI" file: a set of *stanzas*, each describing some discrete element of the character. In turn, each stanza consists of a number of *fields* defining individual attributes. Field names must be unique only within each stanza.

In general, the file contents are plain-ASCII (7 bit) text, with newlines terminating each field. Long lines may be continued by indenting the second and subsequent lines (otherwise field and stanza definitions must *not* be indented at all), but note that the newlines remain embedded in the field data just as they appear in the file. By convention, the character file format uses this for storage of lists of data, putting one list item per line in a multi-line field, while normal single-value fields are generally written out on a single line.

Comments

Lines beginning with "#" or ";" are comments and are completely ignored by GMA. Note that this may be more severe than intended; as the data from the file are loaded and stored, but the comments are *not*, so they will likewise *not* be re-generated when a new data file is written for that character

later. See the `comments` field in the `[title]` stanza (below) for a way to place persistent comments in the data file. GMA uses comment lines beginning with "#" to place information around the data to help hand editing of the files.

Stanzas and Fields

Stanzas begin with the stanza name on a line by itself, surrounded by square brackets. Fields within a stanza consist of a field name and its value, separated by either an equals sign ("=") or a colon (":"). There is technically no difference in either case, but the convention followed by GMA is to use the equals sign for single values, and a colon when introducing multi-line lists of values. Thus:

```
[title]
name=Fred
race=H

[save]
mods:
    +2 vs. cheesecake
    +4 protection from bad puns
```

This shows the definition of the `name` and `race` fields in the `[title]` stanza, and a multi-line field `mods` in the `[save]` stanza.

Throughout this document, we will use the notation "⟨*stanza*⟩:⟨*field*⟩" to refer to a particular ⟨*field*⟩ specifically, or just the field name if that would not be ambiguous.

Value Substitution

Note that this file format allows for a form of value substitution, whereby the value of one field may be interpolated inside the value of another field. If the value of a field contains the pattern: `%(`⟨*name*⟩`)s` then that token is replaced with the value of field ⟨*name*⟩ from the same stanza (or, if no such field exists, from a special stanza named `[DEFAULTS]`). This feature is not used much in GMA, but it is important to note that this means that in order to get a literal percent sign in the field data, it must be escaped (doubled), like so: `description=grants 15%% bonus to die rolls`

SPECIAL DATA TOKEN SUBSTITUTION

Sometimes dynamic data values need to be calculated into text fields on the character record sheet. This can be accomplished by including special data

tokens in fields which support them. These fields are surrounded with curly braces (e.g., "`{str}`" is replaced with the character's strength modifier).

STANZA DESCRIPTIONS

Title Block: [title]

The initial part of the character file determines the basic identity and most essential global attributes for the character go in the `[title]` stanza, in these fields:

`id=`
: Character ID code assigned by GM. This may be of any format but should usually be fairly short and alphanumeric. (Putting something like a GUID or UUID here would just be obnoxious.)

`name=`
: The character's name.

`nickname=`
: A very short form of the character name, if the full name is largish. This will appear on some reports where there isn't a lot of room for full character names. [OPTIONAL]

`player=`
: The name of the human player who is running this character.

`align=`
: The character's alignment. Use standard one-or-two-letter codes here:

Code	Alignment	Code	Alignment
LG	lawful good	NG	neutral good
LN	lawful neutral	N	true neutral
LE	lawful evil	NE	neutral evil
CG	chaotic good		
CN	chaotic neutral		
CE	chaotic evil		

`move=`
: The character's normal movement rate (also known as "speed"). This consists of two space-separated numbers. The first indicates the base speed when unencumbered, and the second when wearing heavy enough armor to impede movement. If not given, the racial default will be used. [OPTIONAL]

`race=`
: The character's race. Use the standard abbreviation for this, which is set by your GM. Typical starting races and their codes include:

Code	Race	Code	Race
D	Dwarf	HO	Half-orc
E	Elf	HG	Halfling
G	Gnome	H	Human
HE	Half-elf		

`sex=`
: Enter `M` for male or `F` for female. [OPTIONAL]

`age=`
: The character's age in years.

`ht=`
: The character's height in centimeters (whole numbers only). If the value is of the form ⟨*n*⟩'⟨*m*⟩'' or just ⟨*n*⟩' where ⟨*n*⟩ and ⟨*m*⟩ are integers, this will be interpreted as a height of ⟨*n*⟩ feet, ⟨*m*⟩ inches. This will be recorded as the equivalent number of centimeters when the file is saved. [OPTIONAL]

`wt=`
: The character's weight in grams (whole numbers only). If the value is expressed in the form ⟨*n*⟩# or ⟨*n*⟩ `oz` (where ⟨*n*⟩ is an integer), then it is interpreted as ⟨*n*⟩ pounds or ounces, respectively. This will be recorded as the equivalent number of grams when the file is saved. [OPTIONAL]

`eyes=`
: Eye color. [OPTIONAL]

`hair=`
: Hair color. [OPTIONAL]

`skin=`
: Skin color. [OPTIONAL]

`appearance=`
: Any other miscellaneous notes about the character's appearance. [OPTIONAL]

`background=`
: Any relevant biographical information. [OPTIONAL]

`deity=`
: Name of the character's deity, if any. [OPTIONAL]

`classes=`
: The class of the character. For multi-classed characters, separate each class code with slashes, so a fighter would be listed as "`F`", but a fighter/cleric/rogue would be "`F/C/R`". The classes available and their codes are set by your `GM`, but the standard starting set include:

Code	Class	Code	Class
A	Alchemist	H	Harrower
AR	Aristocrat	M	Monk
B	Bard	O	Oracle
BN	Barbarian	P	Paladin
C	Cleric	R	Rogue
CA	Cavalier	RN	Ranger
D	Druid	S	Sorcerer
EK	Eldrich Knight	W	Wizard
EV	Evangelist	WC	Conjurer
EX	Expert	WH	Witch
F	Fighter	WM	Wild Mage

`favored_classes=`
: Slash-separated list of class code(s) which are favored for this character. [OPTIONAL]

`levels=`
: The class level(s) of the character. If multi-classed, these are listed with slashes between them and in the same order as the classes listed in the `classes` field.

`cl_mods=`
: If there are any modifications to spell caster levels to be made (e.g. due to a prestige class that raises the caster level in an existing class), use this field to indicate all the adjustments to all levels. This is a slash-separated list just like the `levels` field, but the values give the number of levels to add to the caster level of the corresponding class.

`xp=`
: Current experience total.

`cp=`
: Liquid wealth of the character in copper pieces (whole numbers only). The conversion rate between coins is easy enough to handle this without trouble:

1 pp	1,000 cp
1 gp	100 cp
1 sp	10 cp

`hero_points=`
: The number of hero points the character currently has available. [OPTIONAL]

`capacity_bonus=`
: In some situations, such as when employing a masterwork backpack,

a character's carrying capacity is considered to be higher than normally allowed for their strength. If this field is present, its integer value is added to the character's current effective strength score for the purposes of calculating carrying capacity. [OPTIONAL]

`comments:`
: The value of this field is stored but not printed on the sheet. It may be used for keeping personal notes to yourself about the character. Actually, a field called `comments` may appear in any stanza. Each stanza maintains its own comment field. Upon saving the file from GMA tools, all of the comments from all ability scores are saved in the `[str]` stanza. All others stay in their own stanza.

`notes:`
: This is a large multi-line block of text containing any particular items which need to be detailed for the character. All the text here can be formatted using our markup syntax. Note that blank lines are suppressed in this file format, so to introduce a blank line, put a single vertical bar on a line in the file.

Ability Score Block: [str], [dex], [con], [int], [wis], [cha]

Each ability score (strength, dexterity, and so forth) is described in its own stanza, named for the standard three-letter abbreviation for that score. Eventually, many different fields may appear here to modify the ability scores, but at this point only one is needed. The other effects of or modifiers to these scores are handled automatically.

`base=`
: The base score as rolled, after applying the at-creation racial modifier, if applicable.

`temp=`
: The temporary score which is in effect.

Saving Throw Block: [save]

At least theoretically, each saving throw has its own stanza just like the ability scores do, but this has been effectively dropped in version 4.0 since this information is already inherent in the class and level data. All that remains is the set of overall modifiers to saving throws, which is in the `[save]` stanza:

`mods:`
: A multi-line list of general modifiers to saving throws, one per line. There is room on the standard sheet for two of these to appear (although really that's a bug; it should expand as needed). These are plain text notes to the player; no calculation on these values takes place within the system.

Weapons Block: [weapons], [weapon.*], [mode.*]

Armaments go here. The `[weapons]` stanza lists global attributes for the character's arsenal generally, including:

`active:`
: A list of weapon ID numbers for the weapons which the character is making active use of. These are the ones which appear on the character record sheet's combat section. These values are listed on multiple lines, separated by any whitespace (so multiple IDs may appear on a line as well as multiple lines).

`modes:`
: A list of special attack modes to list on the sheet in addition to the standard set of "Melee," "Ranged," and "CMB,". The meaning of each of these is defined in a `[mode.*]` stanza (q.v.).

Each weapon owned by the character has its own stanza identified as `[weapon.`⟨*id*⟩`]` where the ⟨*id*⟩ is the campaign-unique identifier for the weapon (assigned by the GM). There is a universally unique ID also assigned so this item can be tracked across campaigns but that's too large and unwieldy for everyday use so we use the friendlier one here. These are the IDs listed in the `active` field of the `[weapons]` stanza. Note that there may be more individual weapon stanzas than what are listed on the active list. These are still defined, stored, and tracked, but are considered "out of play" (in storage, perhaps). If a weapon is listed in the `active` field, it *must* be defined in its own stanza.

If you wish to list weapons that don't have GM-assigned numbers, that's fine but they still need a unique (to your character sheet) ID number anyway so you can create a stanza to describe them. In this case, you should make up a descriptive ID with a leading underscore (e.g., "`_claws`") The leading underscore will suppress it from being printed in the ID field on your character sheet.

Each individual weapon stanza, `[weapon.`⟨*id*⟩`]`, contains these fields:

`type=`
: This is a code which identifies the type of weapon described here. These codes are defined by the GM and may be any alphanumeric code. If a weapon is a typical example of its type, this may be the only field necessary to define in its stanza.

`sz=`
: The weapon size code. This is optional, and defaults to "`M`" if not specified. Use these codes for the size:

Code	Size
T	tiny
S	small
M	medium
L	large

If, and only if, any of the following fields are changed from the prototype of this kind of weapon, they are listed as well. These override the standard values for this kind of weapon:

`cost=`
: Market value of the weapon, in copper pieces. [OPTIONAL]

`damage=`
: String description of damage dealt by weapon. This should be in the form "⟨*n*⟩`d`⟨*s*⟩[+⟨*a*⟩]" for each attack (with multiple attacks, such as for double weapons, listed with slashes between them). A comma followed by special attack types may be appended to this. Bonus damage due to magic or character attributes will be added to this value. [OPTIONAL]

`crit=`
: Space-separated list of integer critical-hit damage multipliers. If the `threat` field is given but `crit` is not, then `crit` defaults to the value `2`. More than one value is needed only for weapons which have multiple attack surfaces (double weapons) where each surface has a different multiplier. [OPTIONAL]

`threat=`
: Integer of lowest raw die-roll value threatening a critical hit (i.e., a critical threat of 20—the default case) would be given here simply as "`20`", while a threat of 19–20 would be given as "`19`". If the `crit` field is specified but `threat` is not, then 20 is assumed by default. [OPTIONAL]

`range=`
: Space-separated pair of values. The first is the range increment in feet, and the second is the maximum number of increments allowed, which is typically 5 for hand-thrown projectiles or 10 for device propelled ones. If this field is empty, the weapon will be considered to be a melee weapon. Otherwise, it will be considered to be a ranged weapon. If this field has a range value and `is_melee` is true, then it will be considered to be both a melee and a ranged weapon. [OPTIONAL]

`wt=`
: The weight of the weapon in grams. If this value is in the form ⟨*n*⟩`#` or ⟨*n*⟩ `oz` then it is assumed to mean ⟨*n*⟩ pounds or ounces, and converted automatically to grams.

`damagetype=`

A single value containing any combination of the following letter codes (e.g., pieircing and slashing would be listed as "`PS`"):

Code	Type
`B`	bludgeoning
`P`	piercing
`S`	slashing

`qualities=`

A string containing a set of single-character quality codes which describe the weapon's special properties. Standard weapon types that `GMA` already knows about will already have the relevant qualities defined, so any value given here will override those defaults, not add to them, so override with care. (See also the `add_qualities` field.) Any hyphen characters ("–") in the qualities string are ignored. Otherwise, all other characters must be from the following set:

Code	Quality	Code	Quality
`1`	one-handed	`O`	orc
`2`	two-handed	`R`	ranged*
`B`	bow	`r`	reach
`b`	brace	`S`	nonlethal
`D`	disarm	`s`	simple
`d`	double	`t`	trip
`E`	elven	`U`	unarmed
`f`	fragile	`W`	dwarven
`G`	gnome	`x`	deadly
`g`	grapple	`X`	exotic
`H`	halfling	`Z`	masterwork*
`L`	light	`%`	melee and ranged
`M`	monk		

*Codes `R` and `Z` are deprecated. Now, weapons are considered ranged if they have a range attribute defined for them, and masterwork is noted by setting their `is_masterwork` field.

`add_qualities=`

Like the `qualities` field, but the letters specified are added to the existing set of qualities for the weapon.

`is_masterwork=`

A boolean field. If true, +1 will be added to the weapon's attack bonus.

`magic=`
: If there is an integer value in this field, it will be added to the weapon's attack and damage bonuses. Setting `is_masterwork` is not necessary if it's already got a magic bonus.

`primary=`
: A boolean field for melee weapons, true by default. If true, this weapon is wielded in the primary hand for purposes of calculating the damage bonus.

`offhand=`
: A boolean field for melee weapons, false by default. If true, this weapon is wielded in the off hand for purposes of calculating the damage bonus.

`twohanded=`
: A boolean field for melee weapons, false by default. If true, this weapon is wielded in both hands for purposes of calculating the damage bonus.

`is_melee=`
: A boolean field for ranged weapons, false by default. If true, and there is a `range` value, the weapon is both a melee and a ranged weapon.

`attack=`
: The attack bonus for the weapon. Do not include bonuses due to strength, magic or masterwork.

`name=`
: Descriptive name as appears on the character sheet.

Special Attack Modes

Each special attack mode listed in the `modes` field of the `[weapons]` stanza must be defined in its own stanza which is named `[mode.`⟨*tag*⟩`]` where ⟨*tag*⟩ is the ID tag listed for it in the ⟨*modes*⟩ field. This ⟨*tag*⟩ will be the name of the mode as it appears on the character sheet. It must be a single word but may contain special characters.

These stanzas will each create a block of weapons and attack stats similar to the standard "Melee Attacks" and "Ranged Attacks" sections. For example, you may wish to create such a special attack stanza for a wizard's Hand of the Apprentice attack mode, since that allows the use of certain weapons with different attack modifiers than melee and ranged attacks use, as well as different range values from what is normal for the weapons.

These stanzas may include the following fields:

`ability=`
: The name of the ability score to use for attack rolls instead of the default "`str`". [OPTIONAL]

`sizemod=`
: The size modifier to use instead of the default "`size`". Currently, the only other value for this is "`csiz`", which is what the CMB uses.

`attacks=`
: The set of attack modifiers, if any, in addition to the ability modifier and base attack bonus(es) which apply to this special attack mode.

`active:`
: A list of weapon ID numbers for the weapons to be included in the list for this special attack mode. These can be existing weapons already listed for other attack modes (including the standard melee and ranged attacks), but if you need to change any of their attributes you'll want to create a new weapon entry to customize for this attack mode.

`name=`
: The name of the attack mode as displayed on the character sheet. If this is not given, the ⟨*tag*⟩ will be used.

`is_melee=`
: A boolean field, defaults to true. If true, this attack mode is treated as a melee attack in the sense that damage modifiers for off-hand and two-handed usage are available, and weapon attributes `primary`, `offhand`, and `twohanded` are respected. If false, the attack mode is treated as a ranged attack in the sense that instead of the above, only standard damage is available. If the weapon `qualities` attribute contains the value "`B`", the bow strength penalty, if any, is applied to damage.

Armor Block: [armour], [armour.*], [armour_mode.*]

Protections go here. The `[armour]` stanza lists global attributes for the character's protective equipment in general, including:

`active:`
: A list of armor ID numbers for the pieces of armor which the character is making active use of. These are the ones which appear on the character record sheet's defense section. These values are listed on multiple lines, separated by any whitespace (so multiple IDs may appear on a line as well as multiple lines).

`modes:`
: A list of special defensive numbers in addition to the normal set of "`AC`", "`Flat`", "`Touch`", and "`CMD`". Each of these modes is defined in its own stanza called `[armour_mode.`⟨*tag*⟩`]` (q.v.).

  Each item owned by the character has its own stanza identified as `[armour.`⟨*id*⟩`]` where the ⟨*id*⟩ is the campaign-unique identifier for the item (assigned by the GM). There is a universally unique ID also

assigned so this item can be tracked across campaigns but that's too large and unwieldy for everyday use so we use the friendlier one here. These are the IDs listed in the `active` field of the `[armour]` stanza. Note that there may be more individual stanzas than what are listed on the active list. These are still defined, stored, and tracked, but are considered "out of play" (in storage, perhaps). If a piece of armor is listed in the `active` field, it *must* be defined in its own stanza.

Each individual armor stanza, `[armour.`⟨*id*⟩`]`, contains the following fields. Note that an ⟨*id*⟩ which begins with an underscore ("_") will *not* be displayed on the character sheet. Those ID tags are used for internal identification of effects which are not objects which need an ID number to track them.

`type=`
: This is a code which identifies the type of armor described here. These codes are defined by the GM and may be any alphanumeric code. If an item is a typical example of its type, this may be the only field necessary to define in its stanza. The "`_special`" code is used for things like spell effects where no default protections are assumed and everything applicable needs to be supplied with the following fields.

`sz=`
: The armor size code. This is optional, and defaults to "`M`" if not specified. Use these codes for the size:

Code	Size
`T`	tiny
`S`	small
`M`	medium
`L`	large

`on=`
: A boolean field, where `yes` indicates that the armor or protective device is being worn (or more precisely, actively defending the character). Defaults to `no` if not specified. [OPTIONAL]

If, and only if, any of the following fields are changed from the prototype of this kind of armor, they are listed as well. These override the standard values for this kind of armor:

`cost=`
: Market value of the item, in copper pieces. [OPTIONAL]

`ac_`⟨*type*⟩`=`
: The bonus (as an integer modifier such as "+2") added to the character's armor class, as a bonus of the specified ⟨*type*⟩. Multiple effects or objects providing bonuses of the same ⟨*type*⟩ do not stack with each

other; the largest bonus takes effect. The types available may be defined by the GM, but the standard set inclues:

Code	Bonus Type
armr	Armor
shld	Shield
defl	Deflection
dodg	Dodge

See the comments about stacking bonuses, below. [OPTIONAL]

maxdex=
: Maximum allowed dexterity bonus while wearing this armor. [OPTIONAL]

chk=
: Dexterity check penalty. If the is_masterwork field is true, +1 will be added to this value automatically. [OPTIONAL]

arcanefail=
: Percentage chance (written as a simple integer) of arcane failure.

speed=
: A space-separated list of two values representing the maximum speed for creatures whose default land speed is 30' and 20' respectively.

wt=
: The weight of this item in grams. A value in the form ⟨*n*⟩# or ⟨*n*⟩ oz is assumed to mean ⟨*n*⟩ pounds or ounces (where ⟨*n*⟩ is an integer) and is converted to grams for you.

qualities=
: A string containing a set of single-character quality codes which describe the armor's special properties. Standard armor types that GMA already knows about will already have the relevant qualities defined, so any value given here will override those defaults, not add to them, so override with care. (See also the add_qualities field.) Any hyphen characters ("–") in the qualities string are ignored. Otherwise, all other characters must be from the following set:

Code	Quality	Code	Quality
E	elven	N	natural
G	gnome	o	Orc
H	halfling	s	shield
h	heavy	W	dwarven
L	light	Z	masterwork*
m	medium	$	special

*Code Z is deprecated. Now, masterwork is noted by setting the `is_masterwork` field.

`add_qualities=`
: Just like the `qualities` field, but the letters specified here are added to any existing qualities already defined for the armor.

`name=`
: Descriptive name as appears on the character sheet.

`is_masterwork=`
: A boolean field, defaulting to false. If true, the armor's check penalty is reduced by 1. Armor with a magic bonus is automatically assumed to have this field set to true as well.

`magic=`
: An integer value which gives the enhancement bonus to the armor. This value is added to the armor class. This also implies that `is_masterwork` is true.

Special defensive modes are defined in their own stanzas, named "`[armour_mode.`⟨*id*⟩`]`". Each of these defines a special way the character defends himself or herself beyond the usual armor class and CMD numbers. Usually, these are just different combinations of armor and armor-like spell effects. Each of these modes may include the following fields:

`cmd=`
: If this is defined with a true value, this mode is a variation of the CMD number, with its special combination of values, as opposed to a regular armor class number. [OPTIONAL]

`only:`
: A list of all the armor ⟨*id*⟩ tags for the set of armor and protective effects in force for this mode's calculation. If not given, it is assumed that any armor with the `on` flag is in effect. This overrides ⟨*all*⟩ `on` flags when specified. [OPTIONAL]

`exc_list:`
: A list of defense modifiers to exclude from the calculation. These must be from the set: `armr`, `defl`, `dex`, `dodg`, `natl`, `shld`, and `size`. [OPTIONAL]

`taglist:`
: A list of defense modifiers to include on the sheet, as for the `exc_list` field. [OPTIONAL]

Other field names found in this stanza are retained as attributes to the special defense mode being defined here. See the GMA code for details on how these are used (or bug Steve to document this more).

Health Block: [health]

Information about the character's state of health and healing ability goes here. This includes:

`hp=`
: Total hit points normally available.

`wounds=`
: Number of points of lethal damage taken. (0 for a healthy character).

`nonlethal=`
: Number of points of subdual or non-lethal damage taken. (0 for a healthy character).

`healrate=`
: Healing rate, if any, as a tuple of three space-delimited values: "⟨*n*⟩ ⟨*d*⟩ ⟨*units*⟩", meaning that every ⟨*d*⟩ ⟨*units*⟩ of time, the character regains ⟨*n*⟩ hit points. The value of ⟨*units*⟩ may be `round(s)`, `second(s)`, `minute(s)`, `hour(s)`, `day(s)`, `week(s)`, `fortnight(s)`, or `month(s)`.

`dr=`
: Damage reduction, if any, as a tuple of one or two space-delimited values: "⟨*n*⟩ [⟨*bypass*⟩]", where ⟨*n*⟩ is the number of deflected hit points, and ⟨*bypass*⟩ is the type of damage which bypasses DR (or this is left out if nothing bypasses DR). [OPTIONAL]

`sr=`
: Spell resistance number, if any. [OPTIONAL]

Special Ability Block: [special]

This is one area where there's still a lot of development work to be done, so there's more than the usual number of place holders and temporary hacks which will be refactored in time.

Any special abilities granted by races and classes are generally placed on the character sheet automatically without the need for any explicit mention in this stanza. However, a few of them need information as to how those abilities are specifically selected or configured for this character.

N.B. Some Classes automatically acquire new special abilities as you advance levels. If those require additional information, you must have that information listed here or your character sheet will throw an error when loaded. For example, if you are a cleric, you automatically get the Channel Energy ability, so you *must* have the `C_ChannelEnergy=`⟨*type*⟩ line in your `[special]` stanza for your character file to be valid.

Special abilities and acquired feats are listed in the `feats` field:

`feats:`
: List of feats and abilities, one per line, from the set of defined feats configured into the system.

If the feat name is prefixed with an asterisk ("∗"), that feat will be highlighted as important if the character sheet is printed with the `--highlight-important-areas` option set.

Some special conditions also appear in this section. For example:

`NegativeLevels|`⟨*n*⟩
: The character has ⟨*n*⟩ negative levels in effect. This will automatically adjust the stats printed on the character sheet accordingly.

Additionally, the pseudo-feat code `XXX_SELECT_ONE_FEAT_XXX` may be used to have the sheet indicate that a feat needs to be selected and added in this slot. Likewise, the code `XXX_SELECT_ONE_XXX|`⟨*description*⟩ may be used to indicate that any arbitrary thing still needs to be selected in this place on the character record.

The pseudo-feat code `[Blank Line]` leaves a blank entry in the feat list on the character sheet. This may be useful for visually arranging the feats.

Some class-related special abilities which require some selection on the player's part are listed in the feat section as well.

Note that some of these items require extra parameters. These are separated from the feat/ability code (and from each other if more than one parameter is needed) by vertical bars. Thus, the specification:

```
[special]
feats:
    DraconicBloodlineArcana|fire
    Languages|Common, Draconic, Elven
```

indicates that the character has the Draconic Bloodline Arcana ability, and specifically as it relates to fire energy. Also, the languages spoken include Common, Draconic, and Elven.

If you need to remove a feat or other special ability added automatically by GMA but does not actually apply to this character, list its name in this section with a minus sign in front of the name, e.g., "`-Darkvision`".

Skill Block: [skills], [skill.*]

Each character has a set of skills they can either inherently use or have undergone training to achieve a level of expertise in those areas. The character record file and the printed sheet do not list skills which the character does not have access to, or ones which have no ranks or other character-specific notes. They will be handled automatically.

The `[skills]` stanza lists global values which apply to the skill set of the character generally, including:

`bonus=`
: The number of bonus skill points gained by advancing levels in preferred classes. This is the only part of the character's skill point total which can't directly be calculated.

Each skill which needs to be listed for some reason appears as its own stanza called `[skill.`⟨*name*⟩`]` where ⟨*name*⟩ is the standardized name of the skill. These names are all lower-case and appear as they are listed in the core rulebook. Sub-categories of skills are appended to the base name as "⟨*skill*⟩.⟨*subcategory*⟩" (e.g., "`knowledge.local`").

Inside each stanza, the following fields may appear:

`ranks=`
: Number of ranks of training this character has for this skill. Often this is the only field you actually need to include. [OPTIONAL].

`mods:`
: A temporary kludge allowing for a list of modifiers to apply to the skill. Temporary, because eventually all of these will be automatically applied by whatever ability or object or effect that is causing them. But for now, this is a list, one per line, of modifiers to the skill. Each modifier consists of an effect tag name and a modifier value separated by a vertical bar. (E.g., "`synergy_feat_agility|+2`" means that this skill adds +2 due to the effects of a feat, which goes into the "synergy" box on the character sheet.) See the section on stackable modifier effects below. The character sheet lists three tags separately: "item", "race", "syne" (synergy) , and the sum of all other effects (obeying their individual stacking rules) goes into the "misc" box.

Inventory Block: [inventory]

The character's possessions are listed here in one list field:

`items:`
: The list, one per line, of possessions carried by the player. Each item on the list consists of several sub-fields, separated by vertical bars:

  1. The ID assigned to this item by the GM, if any. May be left blank.
  2. The weight of the item, in grams (integer only). If this has the form ⟨*n*⟩`#` or ⟨*n*⟩ `oz` where ⟨*n*⟩ is an integer, it is interpreted as meaning ⟨*n*⟩ pounds or ounces, which is converted for you into grams.
  3. The location of the item (text string, not interpreted specially). May be left blank.

4. The item's description (text string).
5. For items which come in sets (like ammunition or torches or days' provisions), number of slots to make available for use. These will be drawn on the sheet as dashed outlines of boxes (if there are more of these potential slots than actual used item slots). [OPTIONAL]
6. The number of item slots (subfield 5) which actually represent the number of items you have. This is drawn by a fully-rendered solid-line box on the sheet, which can be crossed out as the items are consumed. [OPTIONAL]
7. The number of used slots (subfield 6) whose items have been consumed. This number of the boxes are drawn with Xs through them. [OPTIONAL]
8. The number of slots whose boxes should be half-checked. Use this however you want, perhaps to indicate that an item has been prepared or is in active use but not consumed yet. [OPTIONAL]
9. The object class for the item. This is one of a pre-set list of values. You can use `GenericItem` as a default if a more specific class is not known or available.
10. A boolean value. Use 0 if the object is not currently being carried, or 1 if it is. This controls whether the object's weight is counted against the character's carrying capacity.

Spell Block: [spells], [spells.*]

Each category of spells available to the character are listed in the `[spells]` stanza, in this field:

`types:`
: A list, one per line, of all the spell types the character can cast. The potential choices for these types are configured into the system by the GM but currently include: `bard`, `cleric`, `fire`, `glory`, `paladin_special`, `sorcerer`, and `sun`. (Obviously, it is a work in progress.)

For each type of spell used, a stanza called `[spells.`⟨*type*⟩`]` is included which gives details about that type of spells for this character, such as which spells they have available to them. The following fields are used in these stanzas:

`all_level=`
: Include all spells possible up to and including the specified level (an integer value from 0 to 9).

`concentration=`
: Manually enter the concentration modifier used when casting spells of this type. If this is not given, it will be calculated based on caster level and relevant ability score modifier. [OPTOINAL]

`except=`
: Exclude all spells which include the given qualifier tag. (E.g., "`except=evil`" would remove all spells which had the "`[evil]`" qualifier on them. Multiple tags may appear here, separated by vertical bars.

⟨*level*⟩`:`
: A list, one per line, of spells to explicitly include by name in that character's list of spells, considering them to be a spell of the given ⟨*level*⟩. The value of ⟨*level*⟩ may be "`sp`" (documentation of a spell-like ability such as a paladin's *detect evil*), "`0`", "`I`", "`II`", "`III`", "`IV`", "`V`", "`VI`", "`VII`", "`VIII`", or "`IX`". The spell names are as shown in the core rulebook, e.g., "`Magic Missile`", "`Bless Water`", or "`Command`".

  Optionally, a spell name may be tagged with the number of instances of that spell prepared or already cast, by giving a set of "`/`" characters to indicate each one prepared, and "`X`" characters for each cast. These flags are separated from the spell name by a colon. Thus, "`Magic Missile:/X`" means that two instances of the *magic missile* spell have been prepared, one of which was already cast. The first two slots are shown along with the main spell stats on the printed character sheet. If 3 or more are needed, an additional line is added to the sheet to give space for them. The characters which may be used here include:

  `/`
  : Spell is prepared as normal. This is noted on the sheet by a single slash through the checkbox.

  `X`
  : Spell has been cast. This is noted on the sheet by a full cross through the checkbox. This may also be entered in lower case (unlike those documented below).

  `d`
  : Spell is prepared as a domain spell.

  `D`
  : Spell was prepared as a domain spell, and was already cast.

  `e`
  : Spell is prepared using the Extend Spell metamagic feat. You will need to note in the spell slot block that a slot one level higher has been taken.

  `E`
  : Spell was prepared using the Extend Spell metamagic feat and has already been cast.

  `i/I`
  : Spell was prepared as Intensified (upper-case if already cast).

  `l/L`
  : Spell was prepared as Enlarged (upper-case if already cast).

`m/M`
: Spell was prepared as Maximized (upper-case if already cast).

`p/P`
: Spell was prepared as Empowered (upper-case if already cast).

`q/Q`
: Spell was prepared as Quickened (upper-case if already cast).

`s/S`
: Spell was prepared as Still (upper-case if already cast).

`v/V`
: Spell was prepared as Silent (upper-case if already cast).

Any other letters may be used here to represent the various metamagic feats, with the upper-case version indicating that the spell was cast and the lower-case one indicating that it was prepared.

Fully named metamagic flags may also be given by enclosing the name in square brackets. If the name is fully lower-case, then the slot is prepared but not yet cast; otherwise it was already cast. For example the slot list "`//m[aquatic][Aquatic]`" means that the spell was prepared five times. Twice normally, once maximized, and twice with the Aquatic Spell metamagic feat. One of the aquatic versions has already been cast.

To indicate the number of spell slots prepared and spent, use the same notation but with the string "`@slots`" in place of the spell name. If domain or specialist slots are included, they are shown as a second set of flags separated from the first set by another colon. Thus, "`I:@slots://X:/`" means that out of the 1st level slots in the spell block for that character, one has already been cast, two more are prepared but uncast, and one domain or specialist slot has been prepared but not cast. Only the `/` and `X` flags are allowed here.

Relationships with NPCs

The `[relationships]` stanza records the current relationship points for each significant NPC with whom the character has a positive or negative relationship.

⟨*name*⟩=⟨*points*⟩
: Each relationship is noted by a field whose name is the NPC's name, and value is the relationship score.

Animal Collection Block: [animals]

To support animal companions, familiars, and the like, the character record file allows the inclusion of monster blocks in a similar manner as is used for the GM encounter sheet files. The overall animal section is introduced by an `[animals]` stanza which has the following field:

`active=`⟨*list*⟩
: where ⟨*list*⟩ is a space-separated list of animal ID values. The IDs used are arbitrary but are probably best left to simple alphanumeric values. Any number of animals may be defined in the file, but only those in this `active` list will be shown on the character sheet.

  Each animal is detailed in its own "`[animal.`⟨*id*⟩`]`" stanza.

`title=`⟨*text*⟩
: The title for the animal section. Defaults to "ANIMALS" but could be explicitly set here to something like "Animal Companions" or whatever is appropriate.

Individual Animal Block: [animal.⟨*id*⟩]

This stanza describes a single animal type that belongs to the character. The full list of fields for creatures is documented below, but note that these were actually designed for generating GM encounter sheets, so not all of them make sense for character files.

Fields marked with * are only supported in encounter sheets ("`.gmo`" files).

Fields marked with + accept a value of "`--`" or "`---`" to indicate that the attribute has no value (such as an undead creature whose constitution score does not exist as opposed to having a numerical value), or for fields like `xp` it indicates that the value taken from the bestiary should not be used, but rather recalculated from the values provided.

The only required field is `species`, which determines what sort of creature this is. This will construct a `Monster` object pre-loaded with all the stats of that creature, so it may be the case that no other fields need to be added. Any other fields will override the base information for that creature type, to provide specific values for that individual creature. GMA is designed to remove fields from the file which are exactly the same as the base creature anyway.

`species=`⟨*name*⟩
: Recall the species information from the GMA monster data. All other fields modify that base information. The ⟨*name*⟩ parameter is spelled as found in the *Pathfinder Bestiary* or other reference materials. The data for monster species are still being added to the system, so ask the GM (or modify the information yourself) if the species you need isn't recognized. Alternatively, you can give the species of a similar creature, and modify it as needed by adding some of the fields listed below. In this case you will likely want to use the ⟨*class*⟩ field to rename the creature to a more correct species name.

`*appearance=`⟨*text*⟩
: A brief description of what the creature looks like.

`ac:`⟨*list*⟩
: Specifies the components which make up the creature's armor class rating. Note that for monsters/animals, the usual calculations which go into armor class are *not* performed, so this needs to be the *complete* list, including size, dexterity, and so forth. The elements of ⟨*list*⟩ appear one per line and have the form ⟨*type*⟩ | ⟨*modifier*⟩ for each of the armor types, which include (but may not be limited to):

Code	Bonus Type
`armr`	Armor
`defl`	Deflection
`dex`	Dexterity
`dodg`	Dodge
`natl`	Natural (hide, scales, etc.)
`shld`	Shield
`size`	Size

: These will be added to the default set of armor class components for creatures of this type. If you wish to clear the existing set and start over, put the special code `@clear` at the start of the list.

`align=`⟨*code*⟩
: The alignment code using the same codes as for characters.

`*armor_misc:`⟨*list*⟩
: This is formatted like the `ac` field, but the codes are the names of AC categories (typically `ac`, `flat`, and `touch`) while the associated modifiers are the amount by which to adjust the final AC of that category. This will be shown as a "misc" modifier to account for other effects on that AC value other than those explicitly called out in the `ac` field.

`attacks:`⟨*list*⟩
: This is a complex field which describes the various attack modes employed by the creature. Each attack mode is listed on its own line in this multi-line field, and consists of the following sub-fields delimited by vertical bars: ⟨*block*⟩ | ⟨*type*⟩ | ⟨*size*⟩ | ⟨*attributes*⟩ | ⟨*ranged?*⟩

  ⟨*block*⟩
  : The creature's attacks are grouped into "blocks" such that on any combat round, it may use one of the blocks. In other words, each block indicates a complete option available to the creature. One block may indicate a single attack, another would show a full-round attack, etc. All of the attacks in the same block have the same value for their block sub-field. It is customary to begin with block 0 and increment by one for each subsequent block. On the character sheet, each block will be drawn with alternating background colors.

⟨*type*⟩
: The weapon or attack mode identifier, which must be a recognized weapon type code such as "`Club`", "`CrossbowHeavy`", or "`UnarmedStrike`". If the creature is using a natural attack mode, the usual ⟨*id*⟩ value to use is "`_CreatureAttackForm`".
  All of the weapon stats for the given ⟨*type*⟩ are pulled from the GMA database and assumed for this attack mode, so all of the information in the ⟨*attributes*⟩ sub-field need only specify deviations from that standard weapon, if any.

⟨*size*⟩
: The size of the ⟨*weapon*⟩ employed. In most cases this will be the same as the creature's size, but it doesn't necessarily need to be. This determines the damage ranges for the weapon. Valid sizes include the standard size category codes for creatures.

⟨*attributes*⟩
: The various weapon attributes are specified here, in one of two forms. If the standard-issue weapon of that type is mostly correct, then you may specify just the changes needed by providing a comma-separated list of "⟨*name*⟩:⟨*value*⟩" pairs, where the ⟨*name*⟩ and ⟨*value*⟩ may be the following:

  `attack:`⟨*mods*⟩
  : The attack modifiers for this attack form as used by the creature. Since monsters use a much-abbreviated combat model, **You must specify all modifiers** used by this mode. (Add the base attack bonus, weapon bonuses, relevant ability modifiers, etc., to get the total(s) shown here. In combat, the number(s) in this field will be all that needs to be applied to the d20 roll.) This value should either be in a form acceptable as input to the die roller (with a leading "d20" added), or a portion of it that is acceptable set in square brackets.

  `cost:`⟨*cp*⟩
  : The cost of the weapon in copper pieces.

  `damage:`⟨*str*⟩
  : A free-form string indicating the total damage for this attack, including all normal bonuses and specials. For example, "`1d6+2, poison`". This value should either be in a form acceptable as input to the die roller, or a portion of it that is acceptable set in square brackets. Any portion of the damage value that should not be repeated for critical hits should be enclosed in angle brackets (e.g., "`6d6 <+1d4 fire>`").

  `crit:`⟨*mult*⟩/⟨*threat*⟩
  : The critical hit stats for this weapon. The ⟨*mult*⟩ value is the multiplier used to increase the number of damage dice, and ⟨*threat*⟩ is the minimum d20 roll which threatens a critical hit.

`damagetype:`⟨*types*⟩
: A single string containing a combination of letters "`B`" (bludgeoning), "`P`" (piercing), and "`S`" (slashing).

`name:`⟨*str*⟩
: The name of the weapon as displayed on the character sheet. If any portion of the name is enclosed in square brackets (e.g., "+1 flaming [sword]"), only the bracketed portion appears on automatic die rolls that are visible to players.

`range:`⟨*distance*⟩`/`⟨*increments*⟩
: For ranged weapons, this specifies the number of ⟨*increments*⟩ and the ⟨*distance*⟩ of each increment.

`reach:`⟨*bool*⟩
: Indicates whether this is a reach weapon.

`wt:`⟨*g*⟩
: The weight of the weapon in grams.

Alternatively, a completely new set of stats can be specified as a simple comma-separated list of values. This includes the fields ⟨*name*⟩, ⟨*attack*⟩, ⟨*damage*⟩, ⟨*crit*⟩, ⟨*range*⟩, and ⟨*reach*⟩, as documented above.

⟨*ranged?*⟩
: A boolean value. If true, this is a ranged attack rather than melee. This prints in a much darker background color than the melee attacks.

`+bab=`⟨*mod*⟩
: Base attack bonus.

`*biased_qty=`⟨*n*⟩
: This functions just like `random_qty`, but it adds ⟨*n*⟩ more instances of the creature with hit points maximized with bias rather than rolling random hp for each. It is the same as adding ⟨*n*⟩ `$` values to the `hp` field.

`+cha=`⟨*value*⟩
: The charisma ability score (not modifier).

`class=`⟨*str*⟩
: A new string to display for the species name. The actual `species` field is used to load all of the stats out of the database, which is normally enough. This field is often used to add things like class levels and other attributes.

`+cmb=`⟨*mod*⟩
: The combat maneuver bonus.

`*cmb_text=`⟨*text*⟩
: Additional qualifiers about the creature's CMB (e.g., "+12 grapple").

`+cmd=`⟨*value*⟩
: The combat maneuver defense value.

`*cmd_text=`⟨*text*⟩
: Additional qualifiers about the creature's CMD (e.g., "42 trip").

`+con=`⟨*value*⟩
: The constitution ability score (not modifier).

`*conditions:`⟨*list*⟩
: This is a list of conditions, one per line, that the creature should have imposed on it when it is spawned into the game. If the pseudo-condition code "`@clear`" appears in the list, the entire set of conditions is cleared at that point. This is useful for replacing the list of conditions defined in the database for that monster rather than adding to it.

`cr=`⟨*value*⟩
: The challenge rating for this creature. This is either a positive integer value or one of the fractions "`1/8`", "`1/6`", "`1/4`", "`1/3`", and "`1/2`".

`+dex=`⟨*value*⟩
: The dexterity ability score (not modifier).

`dice=`⟨*expr*⟩
: Hit dice for the creature. The ⟨*expr*⟩ is a die roll expression such as would be accepted by the `DieRoller.do_roll()` method as documented in `dice`(3). (for example, "`1d8+3d10+12`").

`dict_presets:`⟨*name*⟩`|`⟨*desc*⟩`|`⟨*client_data*⟩`|`⟨*json-obj*⟩
: Defines a set of one or more ephemeral die-roll presets which will appear in the GM's die roller when this creature is in play. This preset, unlike that defined in the `roll_presets` field, is of the `DictPreset` type which defines an arbitrary data structure for the die roller for it to use in some manner that is beyond the scope of this document. The ⟨*name*⟩ and ⟨*desc*⟩ fields respectively provide the name and description for this preset data. The ⟨*client_data*⟩ field holds arbitrary data of interest to any client which consumes this preset. This MUST NOT contain semicolons or vertical bars. The ⟨*json-obj*⟩ field provides the object in the form of a JSON-formatted object enclosed in curly braces. If more than one such dictionary preset is to be defined for this creature, they appear on successive lines in the `dict_presets` field (each after the first indented as usual for this file).

  If it is necessary to break up a long ⟨*json-obj*⟩ value across multiple lines, this is supported. The rule is that successive lines of the `dict_presets` field are gathered one at a time until a valid JSON value can be parsed. At that point, the value is considered to be complete and the next line, if any, will be the start of the next dictionary preset.

If the string `@clear` appears as one of the lines of this field, all existing dict_presets defined for the creature up to that point are cleared. This allows you to either extend or completely replace whatever presets the core monster data had for this creature.

`dr=`⟨*text*⟩
: Damage resistance stats.

`feats:`⟨*list*⟩
: List the feats which the creature has in this block, one per line, in the same manner as for character feats (the feat code, plus any particular parameters needed for that feat, separated by vertical bars). These will be added to the default set of feats for creatures of this type. If you wish to clear the existing set and start over, put the special code `@clear` at the start of the list. If the feat name begins with "`-`", all feats with that name are removed from the creature. Subsequent lines in this field may add new ones in their place.

`+fort=`⟨*mod*⟩
: Fortitude save.

`*generic_name=`⟨*name*⟩[`#`⟨*n*⟩]
: If the true name of the creature is not known to the party at first (such as if the `image_tag` field contains the given name of an NPC they haven't been properly introduced to, or the name of a species they are as of yet unfamiliar with on sight), you can use this field to supply a temporary placeholder name which will be shown on the mapper until the GM chooses to change it to the one given by `image_tag`. If multiple creatures are defined by this stanza, their generic names will have automatic sequence numbers "`#1`", "`#2`", etc., appended to them unless an explicit "`#`⟨*n*⟩" is given here, in which case the sequence numbers will begin with ⟨*n*⟩.

`*hidden=`⟨*bool*⟩
: If this has a true value (such as "`yes`"), the creature token is placed on the map hidden from view of the players. Activating it in the initiative list will not reveal it on the map as would have been the case without this field being set to true.

`hp=`⟨*list*⟩
: Explicit hit points for each creature of this type. If you roll your own hit points for the creature (and this is the most likely case for PC animals), list the current hit points here. If multiple (space-delimited) values are given, then that many individual creatures with these stats will be generated with the corresponding hit point totals. See `random_qty`. Wounds may also be noted in this list by using the following extended syntax for each hit point value in the list:

⟨*t*⟩
: Unwounded creature with ⟨*t*⟩ hit points.

⟨*t*⟩:⟨*w*⟩
: Creature has ⟨*t*⟩ total hit points, out of which ⟨*w*⟩ have been lost due to lethal damage.

⟨*t*⟩:⟨*w*⟩:⟨*n*⟩
: As above, but ⟨*n*⟩ points have been lost to non-lethal damage.

The values for ⟨*w*⟩ and ⟨*n*⟩ may be omitted entirely or specified as `0` to indicate that no such wounds are present, or they may be specified as `X` to indicate that all possible hit points are consumed. Thus, if ⟨*w*⟩ is `X`, the creature is disabled with 0 hp, and if ⟨*n*⟩ is `X`, all the creature's hit points not already taken to lethal damage should be taken as non-lethal.

If the ⟨*t*⟩ value is `@`, this means to roll a random value for the total hit points rather than using an explicit value. Thus,

```
hp=@ @ @
```

is equivalent to

```
random_qty=3
```

except that you can also go on to add wound information such as:

```
hp=@:X @:5 @
```

Likewise, specifying `!` in place of the hit point value means to use the maximum possible hit point value rather than rolling a random one. You can use this wherever you could have used an `@` sign.

Alternatively, specifying `$` in place of the hit point value (instead of `@` or `!`) provides a biased maximum value. This differs from `!` in that it starts with the maximum possible hit point value, but then biases the result ±10%, so the result will be a value between 90% and 110% of the creature's default maximum. (Creatures whose maximum hit point total is less than 10 are simply maximized without applying bias.)

This is intended for game groups which have PCs simply add the maximum number of hit points each level instead of rolling for them. To compensate for this, you can use `!` to maximize the monster hit points too, or `$` to do the same but with some random variation on top to add some individuality and variety to a group of creatures.

Specifying `~` in place of the hit point value causes GMA to take the median value (as if all the dice rolled the median number; e.g., a d8 would be assumed to have rolled a 4).

Any of the above values (⟨*t*⟩, ⟨*w*⟩, and/or ⟨*n*⟩) may be further modified by appending one of the operators `+`, `-`, `*`, or `/` followed by

a number (or the letter `X` to indicate the total hit points not already taken to wounds), to add, subtract, multiply, or divide the value at that point. Thus, `hp=@:4:X/2` rolls a random number of total hit points for the creature, applies 4 hp of lethal damage, and then applies half of the remaining hp as nonlethal damage. These may not be applied to simple numeric values, so `hp=12+5` is not legal.

Floating-point values require a leading digit (i.e., use 0.5 instead of .5).

Multiple such expressions may be given, which are evaluated left-to-right with no order of operations. Thus, `hp=@:4:X*0.6-3` Rolls random hit points, applies 4 hp of lethal damage, and then applies 60% of the remaining hit points, minus 3 hp, as nonlethal damage.

The result of an expression using these operators is constrained to produce a minimum value of 1.

`*image_tag=`⟨*str*⟩

A space-separated list of image tags to be used for the creature token on the map. Usually only one is needed, and will be used for all instances of the creature. If more instances are needed than elements of this field, the last one is repeated as necessary.

The full format of each element is

[⟨*group*⟩::][⟨*image*⟩=][⟨*internal*⟩_]⟨*name*⟩[#⟨*start*⟩][+⟨*alts*⟩]

or

[⟨*group*⟩::][⟨*image*⟩=]@[⟨*culture*⟩,][⟨*opt*⟩:⟨*val*⟩,...,]⟨*gender*⟩[+⟨*alts*⟩]

where ⟨*image*⟩ is the base name of the image file to use for the token (if not the same as ⟨*name*⟩); and ⟨*name*⟩ is the name of the creature as it will appear on the initiative tracker and mapper. If multiple creatures are being defined, they will be numbered as #1, #2, #3, and so forth, unless ⟨*start*⟩ is also specified, in which case that will be the starting number of the sequence instead of 1. If multiple images are defined (as for polymorphs), then ⟨*alts*⟩ is the number of additional images defined.

If the name starts with some alphanumeric characters followed by an underscore, as in "`secret_lizardfolk`", the text up through the last underscore is an internal private descriptor for the creature and will not be displayed to the players or on the map, but it will be visible to the GM in the `gma`(6) program itself. (Note that underscores can be in this internal descriptor, so an image tag of "`very_secret_lizardfolk`" would be displayed as "lizardfolk" to the players and as "very_secret_lizardfolk" to the GM.)

If specifying multiple creatures with different names but identical combat stats, give all of their image tags the prefix ⟨*group*⟩:: to collect

their combat die-rolls into a single generic set rather than needlessly repeating them for each individual.

If the second form is used, a random name suitable to the ⟨*culture*⟩ is generated. See `gma-monster-sheet`(5) for details of the parameters and operation of this directive. For every generated name, the given name is used in place of the ⟨*name*⟩ in the `image_tag`, and the full name is appended to ⟨*names*⟩ (so the name list should not include any static names once the random ones start).

`*imposed_conditions:`⟨*desc*⟩`|`⟨*cond*[,...]⟩`|`⟨*onset*⟩`|`⟨*duration*⟩[`|`⟨*progress*⟩]

Each line of this multi-line field describes a condition the creature may inflict on targets. The sub-fields are:

⟨*desc*⟩
: A description of the effect causing the condition.

⟨*cond*⟩
: The comma-separated list of condition names as defined in the mapper client.

⟨*onset*⟩
: Empty if the onset is immediate, or the time duration before the effect starts.

⟨*duration*⟩
: Empty if the condition does not automatically end, or the time duration after onset that the condition remains in effect.

⟨*progress*⟩
: If non-empty, this names a progress clock type to pre-set in the condition's dialog which will be appropriate for the particular type of poison or other condition being imposed. If you want to further customize the progress clock, an extended syntax may be used in this field which is similar to the contents of the `progress` field (q.v.), and has the form
⟨*typename*⟩`:[*>][`⟨*name*⟩`=][[`⟨*tqty*⟩`[c]][@`⟨*sqty*⟩`]/]`⟨*steps*⟩`:`⟨*state*⟩`=`⟨*color*⟩`,...`
Any of the colon-delimited sub-fields may be empty and unused fields to the right omitted entirely. See the description for the `progress` field below for full details.
If the string `@clear` appears on one of the lines, the existing list of conditions is cleared. Thus, it is possible to either extend the list of conditions in the core species data or to replace it.

`+init=`⟨*mod*⟩
: Initiative modifier. Unlike PCs, this is not a calculated value but is printed on the monster sheet as entered.

`+init_dice=`⟨*spec*⟩
: By default (if the `init_dice` field is not present or has the value `--`

or `---`), a d20 is rolled for each combatant's initiative. If something else is needed, this field provides an alternative die-roll specifier. For example, if the creature has a feat or other ability that allows it to roll two d20s and use the higher of the two for its initiative roll, then `init_dice` would be set to the value "`d20 best of 2`". The modifier given by the `init` field is added to this result, and should not be included in the `init_dice` field.

`+int=`⟨*value*⟩
: The intelligence ability score (not modifier).

`*map_color=`⟨*color*⟩
: The color to use for the threat zone of the creature. Defaults to red if not specified. This can be any color specifier acceptable to Tk widgets, which usually means a standard color name (ala the X11 RGB color list) or an RGB hex value in the form #⟨*rrggbb*.⟩

`*maximized_qty=`⟨*n*⟩
: This functions just like `random_qty`, but it adds ⟨*n*⟩ more instances of the creature with hit points maximized rather than rolling random hp for each. It is the same as adding ⟨*n*⟩ `!` values to the `hp` field.

`*median_qty=`⟨*n*⟩
: This functions just like `random_qty`, but it adds ⟨*n*⟩ more instances of the creature with hit points set at the median value for the creature (based on the arithmetic mean of possible rolls rather than rolling for random hit points). It's the same as adding ⟨*n*⟩ `~` values to the `hp` field.

`move=`⟨*rate*⟩
: The creature's movement rate. For convenience, I like to use the old-school AD&D movement symbology. All movement forms are concatenated together, in these formats:

  ⟨*x*⟩
  : Normal land movement rate of ⟨*x*⟩ feet per round.

  /⟨*x*⟩⟨*c*⟩
  : Flying speed of ⟨*x*⟩ ft/r, with maneuverability class ⟨*c*⟩

Class	Description	Fly Check
`c`	Clumsy	-8
`p`	Poor	-4
`A`	Average	± 0
`G`	Good	$+4$
`P`	Perfect	$+8$

  //⟨*x*⟩
  : Swimming speed of ⟨*x*⟩ ft/r.

`(`⟨*x*⟩`)`
: Burrowing speed of ⟨*x*⟩ ft/r.

`*`⟨*x*⟩
: Speed of ⟨*x*⟩ ft/r in webs.

`^`⟨*x*⟩
: Climbing speed of ⟨*x*⟩ ft/r generally.

`@`⟨*x*⟩
: Climbing speed of ⟨*x*⟩ ft/r in trees.

`name:`⟨*list*⟩
: If the creatures have specific names, list them in this newline-delimited ⟨*list*⟩ (one per line). The same number of names should be given as the number of hit point totals in the `hp` field or the value in the `random_qty` field.

`*poly_gm=`⟨*bool*⟩
: If this field is present and has a true value, then only the GM is allowed to see the polymorph menu for that creature. This allows for shapechangers to appear on the map without the mapper program itself giving away their true natures to the players.

`*poly_sizes=`⟨*list*⟩
: For creatures which are capable of polymorphing or shape changing, this field lists the size categories of each of their defined forms. The number of elements in the list also serves to indicate how many forms are available. Tools such as the `mapper`(6) client will expect to find a separate graphic image for the creature token for each of these forms. For example, a ⟨*list*⟩ value of "`M M L`" means the creature may assume three forms, the first two of which are medium and the thrid is large size. Use of this field makes the ⟨*alts*⟩ portion of the `image_tag` redundant (although you can also use the ⟨*alts*⟩ value in `image_tag` as a note to yourself if you prefer, but we consider the ⟨*alts*⟩ value to be deprecated and it may not be supported in future versions of GMA). Any of the forms acceptable for the `size` field are valid here in each of the list elements.

  Optionally, each size code may end with a colon followed by a comment (e.g., "`M:Human`") which provides a note to the GM about what each of the different forms represents in the game. If the comment begins with an asterisk (as in "`M:*Human`"), then that skin will be the one the creature will use when initially spawned to the map. Only one skin in the `poly_sizes` field may be so designated. The behavior if multiple skins are marked with asterisks is undefined, but the default behavior if none are so marked is to use the first one in the list. Note that the initial skin size code in the list must match the creature's `size` field. (See the BUGS section below.)

  This field is now DEPRECATED. Use `size` instead.

`notes:`

This is a large multi-line block of text containing any particular items which need to be detailed for the creature, not otherwise listed here. All the text here can be formatted using our markup syntax. Note that blank lines are suppressed in this file format, so to introduce a blank line, put a single vertical bar on a line in the file.

`*progress:[*>][`⟨*name*⟩`=][[`⟨*tqty*⟩`[c]][@`⟨*sqty*⟩`]/]`⟨*steps*⟩ | ⟨*typename*⟩ | ⟨*state*⟩`=`⟨*color*⟩ `,...`

Add one or more progress clocks to the creature's "imposed condition" list as a way to track the slow onset or progress of various conditions or afflictions (or other ongoing processes that are involved with the creature during the encounter). Each line in this field defines such a "progress clock" counter. As typical for multi-line lists of this sort, a line consisting of the value `@clear` empties the creature's list of progress clocks defined so far and begins a new list with the following items only.

The progress clock is a convenient way to track a sequence of events over time, such as the current status of a series of saving throws against a creature's venom, where the victim needs to save 6 times, getting 2 consecutive saves for the poison to have run its course. The ⟨*steps*⟩ value gives the number of total steps in the sequence. By default, they must all be completed for the overall progress to be considered finished. If fewer than that number is sufficient to meet the terminal condition, this number is specified by the optional ⟨*tqty*⟩ value. Once that many steps have been completed, the process is considered done. If the steps aren't just binary ("not yet done" vs. "done"), but are something like "not yet done", "failed", and "succeeded" (such as saving throws), and you need ⟨*tqty*⟩ of them to succeed consecutively to count as the terminal condition for the process, then place the letter "`c`" after ⟨*tqty*⟩.

You may start the field with one or both of these special flags:

`*`

Create the progress clock window as soon as the creature is added to the initiative tracker instead of adding it to the conditions menu. This is for processes that will always be in play and don't need to be manually activated.

`>`

Attach the progress clock to the running game clock when it is created so it automatically advances at a rate of one step per round.

You may optionally assign a ⟨*name*⟩ to the progress clock to distinguish it from others in play.

If a number is specified for ⟨*sqty*⟩, that many steps will be pre-filled as a starting number of already-done steps when the clock is created.

If ⟨*typename*⟩ is non-empty, it specifies a progress clock type from your GMA preferences which the other parameters will be modifying.

By default, each step has two possible states: "pending" and "done". If others are needed, you can redefine the set of states in the third field, which is a comma-delimited list of values, each of which is a state name, equals sign, and color code. The first in the list is the initial state each step will assume when the clock is created. The second in the list is the state the clock will automatically flip each step to when it advances (although you can manually flip each step to whatever state you wish by clicking on it individually), and the last one is the terminal state which is the one that marks that the step is successfully completed.

Recalling our example above, a progress clock to track poision saving throws where 2 consecutive saves out of 6 total are required would be specified as so: `progress=2c/6||x=#888888,fail=red,save=green`

Alternatively, if you had a progress clock type called `savingthrow` in your preferences that set up the states and colors for saving throws the way you wanted them, you would just need to use that and specify the number you need: `progress=2c/6|savingthrow`

The difference between adding an entry here vs. the ⟨*progress*⟩ field in the `imposed_condition` field (q.v.) is that here you're saying that the progress clock itself is all you need to track whatever the condition is, e.g., the creature's *ki* pool or some ritual it's carrying out that you want to track; while adding a progress clock to the `imposed_condition` field means that the imposed condition you're describing there, e.g., poison, needs a progress clock to track whatever it is, in addition to whatever else that condition is already doing (e.g., adding a condition status to the target creature).

The values for ⟨*tqty*⟩, ⟨*sqty*⟩, and/or ⟨*steps*⟩ may be either a fixed integer or a valid die-roll expression as described in `gma-dice-syntax`(7) enclosed in square brackets (e.g., "`[1d4+1]`").

`*random_qty=`⟨*n*⟩

As an alternative to the `hp` field, this specifies that there will be ⟨*n*⟩ creatures with these stats whose hit points should be generated randomly (but beware, they will be re-rolled each time the sheet is loaded into the system, although some mitigation is implemented to get consistent die rolls between loads—in essence, if the creature data is not changed between loads, the random die rolls should be the same each time). See `hp`. Including this field is the same as adding ⟨*n*⟩ `@` values to the `hp` field, so if you specify both, they will both take effect, with `random_qty` adding more instances to the set already specified in `hp`.

`*reach=`⟨*ft*⟩
: The space threatened by the creature on the combat grid, in feet. THIS FIELD IS NOW DEPRECATED FOR INPUT. It is now automatically populated from the value in the `size` field, which supports syntax to describe size, space, and reach all in a single value.

`*reach_text=`⟨*text*⟩
: Additional commentary about the creature's threatened area.

`*ref=`⟨*str*⟩
: The page number of the entry for this creature in the reference book or other material it comes from. Usually this is `B`⟨*n*⟩ for page ⟨*n*⟩ of the ⟨*Pathfinder Bestiary*⟩ or `2B`⟨*n*⟩ for page ⟨*n*⟩ from the ⟨*Pathfinder*⟩ ⟨*Bestiary*⟩ ⟨*2*⟩, but could be any of a number of other reference books we use.

`+refl=`⟨*mod*⟩
: Reflex save.

`*regen_hp=`⟨*hp*⟩
: Whereas the `regenerate` field gives a text description of a creature's regeneration, fast healing, or similar ability, this field and `regen_interval` specify in mechanical terms the ongoing standard regeneration effect, if any. This field specifies how many hit points are recovered every interval.

`*regen_interval=`⟨*time*⟩
: The interval at which hit points are recovered when regenerating. This is any time specifier allowed by the GMA game clock.

`*regenerate=`⟨*text*⟩
: List the regeneration ability if any.

`resist=`⟨*text*⟩
: List the resistance stats (asterisks on things it's immune to).

`roll_modifiers:`⟨*name*⟩`|`⟨*desc*⟩`|`⟨*spec*⟩[`|`⟨*varname*⟩[`|`⟨*flags*⟩[`|`⟨*client*⟩]]]

: This defines one or more additional die-roll modifiers and variables. Each line contains the same three fields as for the `roll_presets` field. Additionally, ⟨*varname*⟩ gives a variable name with which the modifier may be added to other die roll specs (usually by specifying it as "`$`⟨*varname*⟩" or "`${`⟨*varname*⟩`}`" in the other die-roll expression, but that is up to the client program performing the die roll). If this is empty, no variable will be created. The ⟨*flags*⟩ parameter gives a set of one-letter flags to control how the modifier works. These may include:

  `e` The modifier should be enabled by default.

`g` The expression before the modifier should be grouped (enclosed in parentheses) before the modifier is added. For example, if a modifier's value is "`*2`" and the die-roll expression you're rolling is "`d20+5`", then without this flag set, the resulting die roll would be "`d20+5*2`", but with the `g` flag it would instead be "`(d20+5)*2`".

The ⟨*client*⟩ field may contain arbitrary client data which is used by whatever system will read this die-roll modifier externally.

The ⟨*var*⟩, ⟨*flags*⟩, and ⟨*client*⟩ fields are optional and may be omitted entirely if they, and all following fields, will not be used.

If the string `@clear` appears as one of the lines of this field, all existing modifiers defined for the creature up to that point are cleared. This allows you to either extend or completely replace whatever presets the core monster data had for this creature.

`roll_presets:`⟨*name*⟩`|`⟨*desc*⟩`[|flags=`⟨*flags*⟩`][|client=`⟨*client_data*⟩`]|`⟨*spec*⟩

This defines one or more additional die-roll presets to add along with those automatically generated for this creature, one preset per line. Each line contains three fields giving the preset name, description, and die-roll specification as documented in `dice`(5). If the string `@clear` appears on one of the lines, the existing presets are cleared. Thus, it is possible to either extend the list of presets in the core species data or to replace it.

In addition to the basic three strings, an optional ⟨*flags*⟩ and/or ⟨*client_data*⟩ field may be added as shown above. The ⟨*flags*⟩ value may be the letter `b` to indicate that by default the die roll should be sent blindly to the GM. The ⟨*client_data*⟩ string is any arbitrary data a consuming client program may wish to use but MUST NOT contain semicolons or vertical bars.

`*size=`⟨*codelist*⟩

The size(s) of the creature, as a space-separated list. In the simplest case, only one code is needed. This describes the size and reach areas of the creature. If the creature has shapechanging ability, specify a code value in ⟨*codelist*⟩ for each "skin" the creature can assume. These may be all different sizes, all the same size, or a combination of sizes.

Each size code must be one of the codes shown in Table C.1, or the general pattern

⟨*category*⟩`[`⟨*natural*⟩`][->`⟨*extended*⟩`][=`⟨*size*⟩`][:[*]`⟨*comment*⟩`]`

where ⟨*category*⟩ is the size category letter, ⟨*natural*⟩ is the natural reach distance and ⟨*extended*⟩ is the extended reach distance (e.g.,

Category	Size	Mod	Space	Reach	Ht/Leng	Weight
f/F	fine	+8	.5′	0′	< .5′	< .125#
d/D	diminutive	+4	1	0	.5–1	.125–1
t/T	tiny	+2	2.5	0	1–2	1–8
s/S	small	+1	5	5	2–4	8–60
m/M	medium	+0	5	5	4–8	60–500
L	large (tall)	−1	10	10	8–16	500–4K
l	large (long)	−1	10	5	8–16	500–4K
H	huge (tall)	−2	15	15	16–32	4K–32K
h	huge (long)	−2	15	10	16–32	4K–32K
G	gargantuan (tall)	−4	20	20	32–64	32K–250K
g	gargantuan (long)	−4	20	15	32–64	32K–250K
C	colossal (tall)	−8	30	30	64+	250K+
c	colossal (long)	−8	30	20	64+	250K+

Table C.1: Creature Size Codes and Their Attributes

with a reach weapon). If ⟨*extended*⟩ is not specified, it defaults to double the ⟨*natural*⟩ distance; the ⟨*natural*⟩ value defaults to the standard distance for the size ⟨*category*⟩. If ⟨*size*⟩ is given, it overrides the creature's token size if for some reason you have a creature, such as the Oliphaunt of Jandelay, which is larger than the largest creature category.

In earlier versions, there were special size categories called `m20`, `M20`, `l0`, and `L0`, which are now just examples of the more general pattern; and `C80`, which needs to be specified as `C80=80` now (but that is a very rare size category).

If the optional ⟨*comment*⟩ field is included, it is used as a note to the `GM` to remind them of what that skin represents. It only appears on the printed encounter sheets. If the ⟨*comment*⟩ begins with an asterisk (as in the size code "`M:*human`"), that skin will be the one that the creature will appear as when first spawned to the map. By default, the first size listed will be the initial appearance.

Note that the multi-value usage of the `size` field supercedes the `poly_sizes` field, which is now deprecated.

`skills:`⟨*list*⟩
: List the skill modifiers for the creature, one per line, in the format ⟨*code*⟩`|`⟨*modifier*⟩. The skill codes are the same list as those used for characters, but only the final resulting modifier is listed here. These will be added to the default set of skills for creatures of this type. If you wish to clear the existing set and start over, put the special code `@clear` at the start of the list.

`*space=`⟨*ft*⟩
: The space occupied by the creature on the combat grid, in feet. THIS FIELD IS NOW DEPRECATED FOR INPUT. It is now automatically populated from the value in the `size` field, which supports syntax to describe size, space, and reach all in a single value.

`*space_text=`⟨*text*⟩
: Additional commentary about the creature's space.

`*spawn=[`⟨*area*⟩`@]`⟨*coords*⟩...
: The coordinates on the map where this creature should appear. If multiple creatures are described in this stanza, this may be a space-delimited list of map coordinates giving the spawn location for each individual in order. If there are more creatures than spawn coordinates, the last coordinate is repeated for the remaining individuals. In this case the GMA tools will arrange these remaining creatures in a block with 5 feet between each.

  If a ⟨*coords*⟩ parameter is prefixed by ⟨*area*⟩`@` then that coordinate set is used only if the creature is encountered in that area ID.

  Each ⟨*coords*⟩ parameter is an alphanumeric string as displayed on the `mapper`(6) program, letter first. For example, "`B12`" refers to the grid square in column `B`, row `12`; likewise, "`8@B12`" refers to the same square, but only if loading the encounter area `8`.

  If a ⟨*coords*⟩ value is enclosed in parentheses (e.g., "`(B12)`" or "`8@(B12)`"), then the token is not immediately put on the map when the encounter is loaded. Rather, the creature will be added to the initiative tracker but with the "skip" status selected. The first time you activate the creature for inclusion in initiative order, it will apprear on the map.

  If the ⟨*coords*⟩ value is a question mark (e.g., "`?`", "`(?)`", "`8@?`", or "`8@(?)`"), then you will be prompted interactively for the coordinates when the creature is placed on the map, so you can wait until then to decide on a suitable location for it to make its entrance.

  If the character "`*`" is given instead of map coordinates, GMA will not attempt to spawn a map token for the creature at all, but will assume that the token is already on the map.

  Creatures may be spawned into locations that are not an integral number of grid lines. This is particularly useful if the creature is less than Small size or to place any creature in an odd location around the existing map terrain. The base coordinate notation (e.g. `AB123`) doesn't allow for this, but the following characters may be added after the letters and/or numbers of the coordinates to adjust the position on a finer resolution:

  `+` Add 1/2 grid distance to the coordinate. Thus, `AB+13` would be right on grid line `13` but half-way between `AB` and `AC`.

> Add 1/4 grid distance to the coordinate. Thus, `AB>14>>` would be 1/4 grid to the right of `AB` and 1/2 grid distance below line `14`.

`spells:`⟨*data*⟩

If a creature has spells or spell-like abilities other than those listed in the creature database already, they can be added here in a specially-formatted field. By default, this *adds* additional classes of spells to the animal. If you want to *replace* the existing list with a new one, start ⟨*data*⟩ with the special line "`@clear`". The spell list is a collection of ⟨*classes*⟩ of spells (Cleric spells, Druid spells, Spell-like abilities, etc.). In each class there may be ⟨*slots*⟩ specified (for prepared spells), and then a number of spell ⟨*types*⟩ (abilities used at will, those used at different frequencies, domain spells, etc.). Finally, within each type is a list of actual spells. Multiple classes may be given, and multiple types per class, but they must appear in the order just described.

Each line in the ⟨*data*⟩ describes a single piece of this data set. These may have the following formats:

∗⟨*class*⟩|⟨*CL*⟩[|⟨*concentration*⟩]

Begins a new class of spells, where ⟨*class*⟩ is a string describing them on the character sheet, ⟨*CL*⟩ is the effective caster level, and ⟨*concentration*⟩ is the concentration check modifier for this class of spells.

<⟨*level*⟩>[⟨*total*⟩|⟨*avail*⟩|⟨*cast*⟩|⟨*prepared*⟩|⟨*normal*⟩[|⟨*known*⟩[|⟨*dcast*⟩|⟨*dprep*⟩]]]

If the current class of spells are prepared, this line defines the spell slots for the specified ⟨*level*⟩. (This value must be entered as a Roman numeral for consistency with how spell level numbers are displayed in GMA.) The ⟨*total*⟩ number of slots is given, or –1 if that level of spell has unlimited usage; then the number of ⟨*avail*⟩able spell slots (normally this is the same as ⟨*total*⟩); ⟨*cast*⟩ is the number of slots already spent; ⟨*prepared*⟩ is the number of slots into which spells have been prepared; ⟨*normal*⟩ is the number of the slots which are not for domain (or other special) slots; and ⟨*known*⟩ is the maximum number of spells known for that level. Optionally, two more values may appear to support use of domain slots: ⟨*dcast*⟩ is the number of domain slots cast and ⟨*dprep*⟩ is the number of them prepared but not yet cast. If none of these values (other than ⟨*level*⟩) are specified, then it's assumed that the level is unlimited. If ⟨*known*⟩ is not present at all, then there is no notion of maximum known spells.

[⟨*title*⟩]⟨*ability*⟩|⟨*mod*⟩

Begins a new type of spells. ⟨*Title*⟩ is the name of the type (or frequency); ⟨*ability*⟩ is the three-letter abbreviation for the relevant ability score; and ⟨*mod*⟩ is the integer modifier for that

ability score. Note that the square brackets around ⟨*title*⟩ are literal characters and are part of the syntax for this line in the file.

⟨*name*⟩|⟨*level*⟩[|⟨*flags*⟩]
: Adds the named spell to the current spell type list in the current class. Optionally, you may also include ⟨*flags*⟩ (showing instances prepared and/or cast as documented above).

`+sr=`⟨*int*⟩
: The spell resistance of the creature.

`+str=`⟨*value*⟩
: The strength ability score (not modifier).

`tables=`⟨*name*⟩|⟨*desc*⟩[|`flags=`⟨*flags*⟩][|`client=`⟨*client_data*⟩]|⟨*spec*⟩ `\n` ⟨*data-rows*⟩...
: If this creature's behavior, spells, abilities, etc., include random lookup tables which must be consulted, these may be encoded into the creature's encounter data sheet using this field. This creates an ephemeral `LookupTable` object in the die roller while the creature is active in combat. The ⟨*name*⟩ and ⟨*desc*⟩ fields give the name and description of the table. The die roll to make when looking up values on the table is given by the die-roll expression ⟨*spec*⟩. The ⟨*flags*⟩ field is optional and, if present, must begin with "`flags=`" followed by a set of one-character flags, which may be:

  `b`
  : By default, send the die rolls blindly to the GM only.

  `m`
  : MA markup codes may be used in the table's text.

  Likewise, the ⟨*client_data*⟩ field is also optional and must begin with "`client=`" followed by arbitrary client data if desired (but MUST NOT contain semicolons or vertical bars), for the consuming client to make use of in some way. Note that vertical bars are allowed in the ⟨*spec*⟩ field as part of the die-roll expression (and since it's the last field on the line they need no special syntax to escape them).

  Following this line, the table itself is defined on the following lines, each indented as usual for this file format. Each line contains two fields, separated with a vertical bar. The first is an integer value which is the maximum die-roll result for which this table entry will be the selected result, and the other field on the line is the text of that table row to be shown to the players and/or GM if that is the roll result. For the final row of the table, the character "`*`" must be used instead of an integer, indicating that this row will be selected for any remaining die-roll result if no previous table row has been selected by that point.

Following that line, if there is another line in the `tables` field, it will begin another table definition as described above. For example, To introduce a lookup table which rolls a d10 to select one of five random actions might be defined as follows:

```
tables=dosomething|random|d10   5|Do one thing   6|Do nothing
8|Make something up   *|Run away   xmorale||flags=m|d%% best
of 2   25|faint   *|attack
```

Note that this table's first row is selected on a d10 roll of 1–5, the next row on a roll of 6, the next on a roll of 7–8, and the last roll on a roll of 9 or more. Following that table there is a second table, "xmorale", which is then defined, which rolls percentile dice twice, taking the better of the two rolls, then deciding the creature will faint if the result was 1–25% on that roll, or attack otherwise.

If the string `@clear` appears as one of the lines of this field, all existing tables defined for the creature up to that point are cleared. This allows you to either extend or completely replace whatever lookup tables the core monster data had for this creature.

`*templates:`⟨*list*⟩

The ⟨*list*⟩ value is a multi-line list of templates, one per line. Each of these will be applied in order to the base creature definition retrieved from the core creature data, before any explicitly overridden values in this stanza are applied. (I.e., all of the other values you specify are overriding the templated creature, giving you the final say in what the attributes of the creature are.)

You may also need to manually adjust feats and other specifics about the creature. (This is where the decision to make monsters' attributes taken as-is from the core data or `.gmo` file without the usual behind-the-scenes calculation begins to lose its appeal, but this will be considered in future feature development of GMA.)

Only the following templates are currently supported by GMA. Note that in each case, a number of parameters may be given (separated by vertical bars) to indicate choices allowed in the application of that template. Since vertical bars may appear in some of these parameters (e.g., regular expressions), a literal vertical bar may be specified as "`{|}`" (which shouldn't appear in a regular expression).

`advanced[|`⟨*xweap*⟩`]`

The `advanced` template increases CR by 1, adds +4 to all non-null ability scores (except for Intelligence $\leq$2), and +2 to natural armor class. As a result, 2 hit points per hit die are added, armor class and saving throws increased, etc. Attack and damage for weapons are increased by +2 due to Strength and Dexterity increases unless the weapon name matches the regular expression ⟨*xweap*⟩.

`fiendish`[|⟨*dr*⟩[|⟨*resist*⟩]]

The `fiendish` template increases CR by 1 if the creature's HD≥5, adds Darkvision 60 feet, DR, energy resistance, spell resistance, and smite good as a special ability. It will do its best to only make changes which are better than the creature's starting capabilities but this may be difficult to do for damage reduction and energy resistances. If necessary, you may specify either or both of these values as parameters, in which case it will simply use those manual overrides in place of computed values. Otherwise, if it is unsure about how to update the existing values, it will add notes to describe the change instead.

`ghost`

This is an acquired template that can be added to any living creature that has a Charisma score of at least 6. A ghost retains all the base creature's statistics and special abilities except as noted in the template. This increases the creature's CR by 2.

`giant`

This template advances the CR by 1, increases the creature's base size by one category, adds natural armor +3, increases damage from attacks by 1 die step; increases Strength and Constitution by +4, decreases Dexterity by –2 (which adds +2 hp per hit die, +2 to Str/Con based rolls, and –1 to Dex based rolls).

`skeleton`[|⟨*HD*⟩[|⟨*cha*⟩[|⟨*hands*⟩[|⟨*variants*⟩]]]]

This template may be applied to any corporeal creature with a skeletal system. Note that the process of applying this to a creature is complex enough that some manual intervention may likely be required.

Parameters:

If GMA can't figure out on its own what the racial hit dice for the base creature should be, enter that value in the ⟨*HD*⟩ parameter; otherwise you may leave it blank.

Normally, a skeleton's Charisma will be set to 10, but some variant forms of skeletons will change that. If so, indicate the desired Charisma score for the skeleton to have that value considered in the calculations performed while applying this template. If left blank, 10 is assumed.

A skeleton gains a claw attack for each hand it possesses. This is assumed to be 2, but if the creature has a different number of hands, including 0, enter that value in the ⟨*hands*⟩ parameter.

The ⟨*variants*⟩ parameter is a comma-separated list of skeleton variants to the basic template. Currently, the supported variants are `bloody` and `exploding`.

The base creature's basic stats change to: NE alignment, undead type, natural AC based on creature size, HD changed to d8 racial

HD only, new saving throws, new CR, new BAB, ability scores; it loses all skill ranks, feats, special attacks and special qualities. It gains Improved Initiative and any (Ex) qualities that improve its attacks.

Note: This template removes many subtypes from the base creature that indicate kind and alignment. If it doesn't get that correct for a particular creature, specify the type explicitly in the .gmo file as an override. Likewise, removing special attacks and the like is difficult since often those are documented in the notes section, so this may also require some manual adjustment. You will also possibly need to alter attack stats due to the change in the creature's other attributes.

`*tmp_hp:`*list*

This entry lists one or more effects giving the creature temporary hit points. Each effect is described on a separate line (as with all such entries in this file, subsequent lines after the first are indented). Each line contains the following vertical-bar-delimited fields:

`tmp_HP` *(int)*

The number of temporary hit points granted. If a part of this value is enclosed in square brackets, it will be rolled as a die-roll specification as understood by the die-roller (see `dice`(3)).

`tmp_damage` *(int)*

The number of wounds already suffered against this temporary hit point total

`expiration` *(absolute or relative time)*

The time remaining until the temporary hit points expire, as a time duration understood by GMA (e.g. "`1 round`").

`description`

This field is optional, but if given, provides a brief description of the source of the temporary hit points.

`type=`⟨*str*⟩

The creature type as listed in the Bestiary. This is a string like "animal" or "humanoid".

`*verbosity=`⟨*level*⟩

This controls the amount of output generated for this creature. This may be useful, for example, if repeating the same kind of monster over and over, so that the full multi-page monster description need not be repeated as well, wasting paper. The value of ⟨*level*⟩ may be one of the following:

`0` The shortest possible format. Only prints information in the stat block which won't follow the standard monster entry,

namely: Templates, Species, Type, Generic Name, Image Tag, Alignment, Reference, CR, XP, and the blocks showing each individual creature's hit points, spell slots and names of prepared spells.

`1` Above, plus the full creature stat block and notes, but not including any spell information.

`2` As above, plus spell names with no descriptions.

`3` As above, plus compressed spell descriptions.

`4` Above, plus full spell descriptions. This is the default.

`vulnerable=`⟨*text*⟩
: Brief list of what the creature is vulnerable to.

`+will=`⟨*mod*⟩
: Will save.

`+wis=`⟨*value*⟩
: The wisdom ability score (not modifier).

`*+xp=`⟨*n*⟩
: The XP this creature is worth. If this is left out, the system will automatically calculate the XP based on the creature's CR. If the special value `--` is given for this field, the XP is recalculated from the CR as if no XP were given at all. This is provided as a way to override the base XP of a creature type which already has an XP value defined.

Administrative Block: [admin]

This is an automatically generated section you shouldn't ever need to mess with.

`GameType=`
: The game rule system identifier. This determines what the entire rest of the file means, which stanzas and fields are recognized, etc. This document describes the file format when `GameType=Pathfinder`.

`CharacterUUID=`
: The universally-unique identifier for this character. This is the character's identity even in other campaigns. The `id` field in the `[title]` is unique only within an individual campaign and should be shorter and more convenient for humans to grok.

`InstanceData=`
: This is an encoded data string the system uses to store information unique to this version of this character's state in the system.

`ModificationTimestamp=`
: The date and time this data was last updated, in ISO format.

`Options=`
: A coded string which records the interactive GMA character GUI tool's saved options for this character.

STACKABLE EFFECTS

In the d20 family of games, skills, character attributes, spells, feats, active magic items, and other effects influence modifiers during game play. The ways in which these effects interact with each other is represented by assigning type names or "tags" to these effects. In the game rulebooks, if the description of an effect says something like, "adds a +2 morale bonus," this is "tagging" that +2 bonus as a morale effect, which means it will not "stack" with other morale bonuses but will stack with other types of bonuses such as luck bonuses or dodge bonuses.

When more than one non-stackable bonus is in effect at the same time, the highest one counts, overshadowing the others. If multiple stackable bonuses appear, their bonuses are summed and all apply.

But as with any general rule, there are exceptions as noted in the rulebooks, so sometimes effects with the same name *can* stack with each other. The way this software handles these rules, as you'll note when mentioning effect tags in this data file, is to use a four-letter tag code for each effect in the game (e.g., "`armr`" for armor protection bonuses). All effects whose tags are exactly identical do not stack; the highest will be used. If there were three effects in play:

```
mora +2
mora +1
mora +4
```

then the total "`mora`" (morale) bonus is +4.

However, non-identical tags which are equal to the first four characters are stackable and are added together:

```
morale_a +2
morale_b +1
morale_c +4
```

These would add to a "`mora`" bonus of +7. (The overall effect type is a four-character tag; the extra characters are used just for allowing this stacking effect to be handled.)

A more complex example:

```
race_keensenses +2
race_affinity +1
race_artifact +5
race_artifact +2
race +3
race +1
```

This results in a "race" modifier to whatever ability or check is being described totalling +11. The two racial modifiers due to artifact do not stack, so we ignore the +2 and only keep the +5. That +5 does, however, stack with the affinity and keensenses sources of this racial modifier. The not-otherwise-qualified racial modifiers of +3 and +1 don't stack with each other, so only the +3 contributes to the total. So that's 2+1+5+3=11.

SPECIAL TYPOGRAPHY EFFECTS

The following special character sequences are translated when generating to get special characters on the character sheet output as shown in Table C.2.

SEE ALSO

gma-monster-sheet(5), gma(6), printchar(6).

HISTORY

This describes the 4.1 version of gma-character-record's data file format. A slightly different format was used in version 3.x, and an utterly different format was used in earlier versions.

AUTHOR

Steve Willoughby / steve@madscience.zone

BUGS

This *really* needs to be wrapped in a GUI. (It's not that hard to work with this format, but it's tedious.) The GUI tool has been started, but isn't complete yet.

There should be allowance for more than 2 saving throw modifiers.

There is no way to override the critical hit damage for a weapon to make it *not* threaten critical hits at all. There are currently no known weapons which really work like this, however.

The skill system needs a way to list special uses of skills which inherit modifiers from the base but add more specific ones.

`+/-`	plus-minus symbol (±).
`x`	(when preceded and/or followed by digits) multiplication sign (×).
`[/]`	division sign (÷).
`-`	(when followed by digits) minus sign (−); otherwise it's a hyphen (-).
`---`	Em-dash (—) (or, if that's not possible, two en-dashes (–– are used.)
`--`	En-dash (–)
`[S]`	Section symbol (used in damage specifications to indicate non-lethal, a.k.a. subdual, damage) (§).
`[1]`	Superscripted 1 (^{1}).
`[2]`	Superscripted 2 (^{2}).
`[3]`	Superscripted 3 (^{3}).
`1/2`	(when not surrounded by other alphanumeric characters) fraction $^1/_2$.
`1/4`	(when not surrounded by other alphanumeric characters) fraction $^1/_4$.
`3/4`	(when not surrounded by other alphanumeric characters) fraction $^3/_4$.
`^o`	degree sign. (°).
`[c]`	copyright symbol (©).
`[R]`	registered trademark symbol (®).
`AE`	Æ ligature.
`ae`	æ ligature.
`[<<]`	double angle open-quote («)
`[>>]`	double angle close-quote (»)
`^.`	centered dot (not necessarily a heavy bullet) (•).

Table C.2: Typography Effects

The special calculated values ({...}) are named inconsistently. This needs to be cleaned up. (UPDATE: A more consistent set of names has been created but the old legacy names retained for the time being. See the API documentation for details.) Also, when things like spell descriptions use these, it is harder to see where the variables are (since they may need to change due to metamagic effects).

A future release may do away with calculated values entirely, since they've proven to be more of a problem than a help in a lot of cases where GMA either can't know values like caster level to factor into them, or the calculated values are invalid due to ongoing battlefield conditions, but the use of the calculated values obscures how they're to be (re-)calculated.

When comments in the ability score stanzas are found, they are all moved to the `[str]` stanza when a GMA tool saves the character file again.

The code explicitly bothers to do this on purpose, but at this point I don't remember the rationale behind that decision.

There is some fragility in some of the input fields, such as the `client_data` and other fields in die roll preset objects which this file just assumes you're going to play nicely and not put anything in there that will interfere with the internal encoding of those objects without actually knowing what that encoding is. Apart from the general advice of using alphanumeric values for `client_data` fields (or always base-64 encoding them) and sticking in general to simple alphanumeric data for any data field unless told otherwise, the monster module could use a bit more input validation when reading this file.

gma-dice(5)

NAME

gma-dice – GMA Die Roll Preset Save File Format

DESCRIPTION

The version 2 file format for GMA's die roll presets is described in this document.

The file is UTF-8 (of which plain-text 7-bit ASCII is a subset). The first line of the file MUST have the format: `__DICE__`:⟨*version*⟩ [⟨*datalist*⟩] where ⟨*version*⟩ is the map file format version this file conforms to. While readers should try to read any file they are given, they may not be able to understand anything in a file with a higher version number than they support, or with a version number lower than they are prepared to offer backward compatibility. Such a program MAY refuse to read the file.

Following this are a series of records which MAY appear in any order. Each record has the form

«⟨*type*⟩» ⟨*JSON*⟩

where ⟨*type*⟩ is the record type (which may be `__META__` or `PRESET`) and ⟨*JSON*⟩ is a JSON-formatted data record, which may continue for multiple lines as described in `gma-mapper`(5). JSON field names are case-sensitive. Their values are strings unless otherwise noted.

The characters « and » are Unicode U+00AB and U+00BB respectively.

The final line of the file MUST be exactly

«`__EOF__`»

Metadata records (`__META__`) are JSON objects with the following fields:

`Timestamp` *(int)*
: Unix timestamp indicating when the file was generated or last edited.

`DateTime`
: Human-readable version of the timestamp in any format. Software *never* uses this value. Programs MUST only use the `Timestamp` value.

`Comment`
: A comment about this file, if desired.

`FileVersion` *(int)*
: When reading a file, this field is populated with the input file's format version number. It is ignored if present when writing an output file.

Die-roll preset records (`PRESET`) are JSON objects with the following fields:

`Name`
: The name by which this preset is identified to the user. This MUST be unique within the set of presets defined in this file. (More generally,

the server requires that each preset have a unique name within the list of presets defined for each user of the service.)

`Description`
: A text description of the purpose of the preset.

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

EXAMPLE

```
__DICE__:2
«__META__» {
   "Timestamp": 1570643001,
   "DateTime": "Wed Oct 09 10:43:21 PDT 2019"
}
«PRESET» {
   "Name": "attack",
   "DieRollSpec": "d20+5"
}
«PRESET» {
   "Name": "damage",
   "Description": "My usual damage roll",
   "DieRollSpec": "2d6+1d5 fire"
}
«__EOF__»
```

SEE ALSO

`gma-dice`(3), `gma-mapper`(5), `gma-mapper`(6).

HISTORY

This facility appeared in the 3.31.2 mapper client.

Version 2 of the file format switched from TCL to JSON data structures.

AUTHOR

Steve Willoughby / steve@madscience.zone;

BUGS

The current implementation of file reading code in GMA will accept input more permissively than this spec requires, strictly speaking (e.g., an otherwise valid JSON object where the final brace is not by itself on a line), but

this behavior should not be construed as a requirement for other code. File readers and writers should follow this specification as the authoritative standard.

gma-mapper(5)

NAME

mapper – battle grid save file format

DESCRIPTION

The version 23 file format for the `gma-mapper`(6) battle grid program is described in this document.

The file is UTF-8-encoded Unicode, of which 7-bit ASCII is a subset.

The first line MUST have the form

`__MAPPER__`:⟨*version*⟩

where ⟨*version*⟩ is the map file format version this file conforms to. Following this are a number of data records of the form

⟨*type*⟩ ⟨*object*⟩

The ⟨*object*⟩ is a JSON-formatted data structure appropriate for the ⟨*type*⟩ of value being described by this record. The line MUST begin with the ⟨*type*⟩ designation. The JSON-formatted ⟨*object*⟩ data must either be completely contained on one line, or be continued over multiple following lines, all of which SHOULD be indented by at least one space or tab. A line MUST NOT begin with the character « unless it is the start of a new record or the EOF marker (i.e., this character in the first position on a line marks the start of a new record).

The delimiter characters « and » are Unicode codepoints U+AB and U+BB respectively.

JSON field names are case-sensitive. Their values are strings unless otherwise noted.

The final line of the file MUST be the text «`__EOF__`» to indicate the end of the data.

The following data records MAY be present:

«`__META__`»
: File metadata. The JSON object may contain the following fields:

  `Timestamp`⟨ *(int)*⟩
  : Modificaion time as a number of seconds from the epoch.

  `DateTime`
  : The modification time as a human-readable string value in any form. (Software always uses the ⟨*Timestamp*⟩ field; this is only for the convenience of human readers.)

  `Comment`
  : Random comments about the file.

  `Location`
  : The name or description of the location described by the file.

`FileVersion`⟨ *(int)*⟩
: When a file is read, this field is populated with the input file's format version number. It is ignored if present when writing a map file.

«⟨*element_type*⟩»
: Defines a map element using a JSON structure identical to the JSON data defined in the mapper protocol in `gma-mapper`(6) for the `LS-`⟨*element_type*⟩ command. This includes ⟨*element_type*⟩s `ARC`, `CIRC`, `LINE`, `POLY`, `RECT`, `SAOE`, `TEXT`, and `TILE`.

«IMG»
: Declares the server ID by which a named image may be retrieved by the client (or a local pathname if the file is local) using a JSON data structure similar to that of the `AI` server command. See the example below.

«MAP»
: Declares the existence of a map file which may be retrieved from the server using a JSON data structure identical to that of the `L` server command, except that only the ⟨*File*⟩ field is used.

«CREATURE»
: Places a creature token on the map using a JSON structure identical to that used by the `PS` server command.

COMPATIBILITY

Programs are expected to silently ignore data fields which they don't expect, and to default any missing fields with the zero value for their data type. Thus, it is not always necessary for the file version number to increase if a change in data fields would produce a benign default behavior given these assumptions.

EXAMPLE

```
__MAPPER__:23
«__META__» {
  "Timestamp": 1650611775,
  "DateTime": "Fri Apr 22 00:16:15 PDT 2022",
  "Location": "Room 12 of the Dungeon"
}
«LINE» {
  "ID": 1,
  "Fill": "#00e400",
  "X": 425,
  "Y": 107,
```

```
  "Z": 1,
  "Points": [
    {"X": 433, "Y": 337},
    {"X": 553, "Y": 392},
    {"X": 600, "Y": 243},
    {"X": 424, "Y": 108}
  ],
  "Width": 5
}
«RECT» {
  "ID": 2,
  "Line": "black",
  "Fill": "#00e400",
  "Width": 5,
  "X": 593,
  "Y": 122,
  "Z": 2,
  "Points": [
    {"X": 807, "Y": 443}
  ]
}
«IMG» {
  "Name": "re5_e2",
  "Sizes": [
    {"File": "SJy9rpbTDElsCfklYHMvsgre5_e2@50", "Zoom": 1.0},
    {"File": "SJy9rpbTDElsCfklYHMvsgre5_e2@100", "Zoom": 2.0}
  ]
}
«__EOF__»
```

This file contains two objects:

#1 is a line in the shade of green with (R=0, G=0xe4, B=0), with line segments of width 5 going between points (425,107) – (433,337) – (553,392) – (600,243) – (424,108).

#2 is a black rectangle filled in with the same shade of green as line #1, with a line width of 5, between opposing corners (593,122) and (807,443).

Additionally, it indicates that an image called “re5_e2” may be found with server ID “SJy9rpbTDElsCfklYHMvsgre5_e2@50” at zoom factor 1.0, or with server ID “SJy9rpbTDElsCfklYHMvsgre5_e2@100” at zoom factor 2.0.

SEE ALSO

`gma-mapper`(6), `gma-rendersizes`(6).

FILES

The exact location of cache files may vary depending on the operating system platform. For example, on macOS the `cache` directory is found in `~/Library/Caches/gma-mapper`. On Linux and FreeBSD it is located in `~/.caches/gma-mapper`. On Windows it is located in `%LOCALAPPDATA%\gma-mapper`. If no specific cache directory can be determined, `~/.gma/mapper/cache` is used by default.

`~/.gma/mapper/cache/`⟨***name***⟩`@`⟨***zoom***⟩`.gif` The GIF-format file for image with the given ⟨*name*⟩ and scaled to the given ⟨*zoom*⟩ factor.

`~/.gma/mapper/cache/`⟨***id***⟩`.map` The map file with server ID ⟨*id*⟩.

HISTORY

This describes the 4.x versions of `mapper`.

Changes as of File Format 2

Version 2 files differ from version 1 files in that they add the `JOIN`, `SPLINE`, `START`, `EXTENT` and `ARCMODE` attributes, the `arc` object type, and the "no-fill" (empty string) mode for the `FILL` attribute.

Changes as of File Format 3

This file retains compatibility with the version 2 format, with the following differences.

Each object now has a ⟨*z*⟩ coordinate in addition to the (⟨*x*⟩,⟨*y*⟩) coordinates of its reference point as in previous versions. The ⟨*z*⟩ coordinate indicates the display "stacking order" of objects on the screen. Each object is "flat" and are rendered in order from the smallest to the largest value of ⟨*z*⟩ so that the lowest ⟨*z*⟩ value is at the "bottom" with the others overlapping on top of them. These serve only to indicate the relative positioning of objects. There is no absolute meaning to any particular value for ⟨*z*⟩.

Added the `HIDDEN`, `LEVEL`, and `GROUP` object attributes.

Changes as of File Format 4

This format is compatible with File Format 3, except that it adds the following attributes to support tile objects:

`LAYER` The layer attribute may now include the value `tiles` which is rendered below everything else, including the grid lines.

`IMAGE` This attribute appeared for the first time in this version.

`SIZE` This attribute appeared for the first time in this version.

Programs which read newer version files should deal correctly with older files which do not contain the newer elements.

Changes as of File Format 5

This introduced images, creatures, and spell areas of effect.

Changes as of File Format 6

The image definition syntax was extended to allow externally downloaded image files.

Changes as of File Format 7

Additional creature attributes were added, such as `NOTE` and `SKIN`.

Changes as of File Format 8

Added text objects. This includes the addition of the `ANCHOR`, `FONT`, and `TEXT` attributes.

Changes as of File Format 9

Added `ARROW` and `DASH` attributes. For backward compatibility with format 8, if either of these are missing from the file, defaults are assumed.

Changes as of File Format 10

Added `ELEV` and `MOVEMODE` attributes.

Changes as of File Format 11

Added `HEALTH` attribute.

Changes as of File Format 12

Added `F` record type.

Changes as of File Format 13

Added the optional ⟨*blur*⟩ element of the `HEALTH` attribute. Removed the `SELECTED` attribute. This is a transient state used during the mapper program's operation and has no point being saved with the object. This attribute was moved to the private name `_SELECTED` instead.

Changes as of File Format 14

Added the `STATUSLIST` attribute.

Changes as of File Format 15

Added the `SKINSIZE` attribute.

Changes as of File Format 16

Added the LOCKED attribute.

Changes as of File Format 17

SIZE only applies to creature tokens. The version 16 document incorrectly specified that map elements (notably tiles) also had a SIZE attribute but this was never implemented. Instead, they had unofficial attributes _BBHEIGHT and _BBWIDTH which have now been promoted to official status as BBHEIGHT and BBWIDTH.

File Formats 18 and 19

These are reserved for use in case map format changes are needed before moving to map version 4.x.

Changes as of File Format 20

This version introduced the JSON-based file format, replacing the older key/value line sequence.

Changes as of File Format 21

The fields for creatures have been changed as documented in gma-mapper-protocol(7).

Changes as of File Format 22

Animation support introduced in protocol 407 alters the format of the IMG records in this file correspondingly.

AUTHORS

Steve Willoughby / steve@madscience.zone; Elevation and movement modes added by John Mechalas.

BUGS

The LEVEL, GROUP, and LAYER attributes are not actually implemented yet in the mapper program. They're documented here and will be accepted by the mapper but none of their behavior documented above will actually happen.

The token object type is deprecated and clients are free to be unaware of its existence.

The current implementation of file reading code in GMA will accept input more permissively than this spec requires, strictly speaking (e.g., an otherwise valid JSON object where the final brace is not by itself on a line), but this behavior should not be construed as a requirement for other code.

File readers and writers should follow this specification as the authoritative standard.

Changes as of File Format 23

Map elements may now have a `Stipple` attribute.

gma-monster-sheet(5)

NAME

gma-monster-sheet – data file format for encounter run sheets

SYNOPSIS

This document describes the file format GMA uses to store monster information. These files typically have a "`.gmo`" filename suffix.

DESCRIPTION

See the `gma-character-record`(5) entry for the basic information about how this file is structured. This has the same format, but with different stanzas.

The run sheet is organized like the adventure scenarios are: by module, with each module containing areas, and each area containing monsters.

The `[module]` stanza defines attributes of the module described by the GMO file. Only one module may be described in a single file. This stanza has the following fields:

`title=`⟨*name*⟩
: The module name, which will be printed at the top of the sheet(s).

`id=`⟨*id*⟩
: The module ID number.

`areas=`⟨*list*⟩
: A list of area IDs, separated by any whitespace. (This may include continuation to subsequent lines as described in `gma-character-record`(5).) The IDs in the ⟨*list*⟩ may be arbitrary.

`culture=`⟨*name*⟩`[,`⟨*opt*⟩`:`⟨*val*⟩`,`⟨*opt*⟩`:`⟨*val*⟩`,...]`
: Set the default culture for all creature names to be generated in this module. The given ⟨*name*⟩ must be one of the available cultures defined in the system. The list may be obtained by running `namegen`(6) with the `--list-cultures` option, or via the API by calling `all_culture_names()` from the `PFNameGenerator` module. If specified, each ⟨*opt*⟩`:`⟨*val*⟩ pair sets name generator parameters as desired. These may be:

  `start:`⟨*text*⟩
  : All names are generated such that they begin with ⟨*text*⟩.

  `lmin:`⟨*n*⟩
  : Set the minimum name length to ⟨*n*⟩.

  `lmax:`⟨*n*⟩
  : Set the maximum name length to ⟨*n*⟩.

  `surnames:`⟨*bool*⟩
  : Generate surnames along with the given names if ⟨*bool*⟩ is nonzero.

`if:`⟨*bool*⟩
: For image tags only, show only the first name even if a full name is generated.

Note: Random elements in this file, including generated names and hit points, are calculated using a random number seed derived from the module ID, area ID, and monster ID. This means they will be consistently generated every time the file is printed or loaded, but changing these IDs will cause them to change to new values. They may also change due to edits to other parts of the monster, such as number of individuals, hit dice, etc., since that changes how many dice are rolled while the creature is being created.

Areas

Each area ID defined by the `areas` field in the `module` stanza is defined in its own stanza called `[area` ⟨*id*⟩`]` (note the single space in the stanza name). This stanza includes the following fields:

`title=`⟨*name*⟩
: The name of the area.

`monsters=`⟨*list*⟩
: A whitespace-delimited list of monster IDs describing what encounters are located in this area. These may be continued on multiple lines since any whitespace delimits the elements of ⟨*list*⟩. These IDs are arbitrary and may be reused in other areas if identical encounters exist in more than one place. The special ID string "`//`" forces a page break at that point.

`extra_monsters=`⟨*list*⟩
: A whitespace-delimited list of spawn descriptors, each of which causes a creature token to be spawned onto the map when the area is loaded into GMA. Unlike the `monsters` field, these creatures are not fully described, nor are they tracked in the initiative list. No health information is created or tracked for them, either. They are simply there, in the background, doing background things, in the same fashion as extras on a movie set.

  Each spawn descriptor in the list is a multi-field string with each field separated by vertical bars. In other words, they have the form:

  ⟨*image_tag*⟩ | ⟨*size(s)*⟩ | ⟨*color*⟩ | ⟨*spawnpoint(s)*⟩

  These fields have the following meanings:

  ⟨*image_tag*⟩

[⟨*image=*⟩][⟨*internal*⟩_]⟨*name*⟩[#⟨*seq*⟩]
[⟨*image=*⟩]@[⟨*culture*⟩,][⟨*opt*⟩:⟨*val*⟩[,...]]⟨*gender*⟩

This has the same definition as the field of the same name for monster definitions, as explained in `gma-character-record`(5), except that the ⟨*group*⟩:: prefix is not allowed. This indicates the name to be displayed on the creature's nameplate as well as the graphic token image to show for it.

Currently, the second form (automatic name generation) is not implemented, but it may be in a future release.

⟨*size(s)*⟩

⟨*category*⟩[⟨*natural*⟩][`->`⟨*extended*⟩][=⟨*size*⟩][:⟨*comment*⟩][,...]

This specifies the size of the creature's token just as the `size` field does in monster definitions, with the addition that you may specify multiple size values separated by commas. Doing so indicates that the creature has multiple "skins" which may each be different sizes. Thus, this field also takes the role of the `poly_sizes` field of a monster definition stanza.

⟨*color*⟩

This field is the same as the `map_color` field in monster stanzas. It controls the background color of the creature's nameplate as well as the shading of the creature's threat zone if somehow it got introduced into combat.

⟨*spawnpoint(s)*⟩

This is a comma-separated list of map coordinates indicating where the creature's token is to be placed on the map. If more than one coordinate value is given, that indicates that multiple creature tokens should be placed (one per coordinate value). Each value is an alphabetic column name and numeric row number such as "`B16`". If a coordinate is just a question mark (e.g., "`?`"), the `GM` will be prompted for a map location when the area is loaded. If a coordinate is enclosed in parentheses (e.g., "`(AA42)`" or "`(?)`"), then it will be placed on the map hidden from player view. The `GM` will need to remove the "hidden" pseudo-condition from the token to make it visible to the players.

Alternatively, if a spawn descriptor is simply the string "`//`", then a page break is forced at that point in the list when printing the encounter sheet.

`culture=`[⟨*name*⟩,][⟨*opt*⟩:⟨*val*⟩,⟨*opt*⟩:⟨*val*⟩,...]

As with the same entry in the `[module]` stanza, this sets area-specific default values for name generation. Monsters in this area will use these defaults.

Monsters

Each monster listed in any `area` stanza must be defined in its own monster stanza, named `[m` ⟨*id*⟩`]` (note the single space in the stanza name). The primary fields needed in this stanza are:

`species=`⟨*code*⟩
: Where ⟨*code*⟩ is a creature code defined in GMA, this defines the creature to be encountered here.

`name:`⟨*list*⟩
: A newline-delimited list of creature names. This is an optional field but if given it assigns personal names or other descriptive text to individual monsters. Note that blank lines are not removed from ⟨*list*⟩ so the first name must be on the same line as the `name:` field name. [Optional]

At least one of the following must be given to indicate the individual instances of monsters of the given species:

`biased_qty=`⟨*n*⟩
: Adds ⟨*n*⟩ creatures with hit points biased around the maximum, as if ⟨*n*⟩ `$` characters were added to the `hp` field.

`maximized_qty=`⟨*n*⟩
: Adds ⟨*n*⟩ creatures with maximum hit points, as if ⟨*n*⟩ `!` characters were added to the `hp` field.

`median_qty=`⟨*n*⟩
: Adds ⟨*n*⟩ creatures with the arithmetic mean hit points for their kind, as if ⟨*n*⟩ `~` characters were added to the `hp` field.

`random_qty=`⟨*n*⟩
: Generates ⟨*n*⟩ creatures, with randomly-rolled hit points each, as if ⟨*n*⟩ `@` characters were added to the `hp` field.

`hp=`⟨*list*⟩
: A whitespace-delimited list of hit point values. One creature will be generated per element of this list, with the explicitly-listed hit point value. See the `hp` field description in the `[animal.`⟨*id*⟩`]` stanza of `gma-character-record`(5) for details about the values which can appear here.

If both `hp` and `random_qty` (and/or the other random generation fields listed above) are given, the explicitly-specified hp values will be taken for the first instances of the creature type, and then ⟨*n*⟩ more creatures will be added with randomly-rolled hit points. Thus,

```
random_qty=3
hp=12 43
```

is equivalent to

```
hp=12 43 @ @ @
```

If this monster is described already in another section, as would be the case if mostly-identical creatures appear in multiple places but aren't stock species pulled from the GMA core data, you can avoid repeating the monster block over and over by making each additional one be "based on" the one detailed entry.

This is done by adding the field:

`based_on=`⟨*entry*⟩
: Indicates that this monster entry should include all of the fields from the monster in section "`[m` ⟨*entry*⟩`]`". Any fields in this entry will override those included from the referenced entry. The referenced entry may itself include a `based_on` field.

Any additional customizations needed over the base creature known to GMA may be performed in the same fashion as for the `animal` stanzas of the character record sheet (q.v.).

SEE ALSO

`gma`(6), `gma-character-record`(5), `printmon`(6).

HISTORY

This describes the 4.1 version of `gma-monster-sheet`'s data file format. A very different format was used in earlier versions.

AUTHOR

Steve Willoughby / steve@madscience.zone

BUGS

Needs a way to easily apply scale factors and templates.

gma-style.conf(5)

NAME

style.conf – GMA GUI Style Configuration File

DESCRIPTION

This file is now deprecated. Its function is now superseded by the in-app preference editor.

The `gma-mapper`(6) battle grid program, along with other GUI tools such as `gma-initiative-clock`(6), use colors and fonts to indicate a number of things on their displays. The `style.conf` file described here allows the user to customize any or all of these settings. This file may be specified to the application using command-line options. Otherwise, if the file `~/.gma/mapper/style.conf` exists, it will be read by default. See the manpage entries for the individual programs for details.

The overall format of this file is that of a classic Windows INI file; that is, the file is divided into named stanzas. Within each stanza is a set of named values, one per line.

Each stanza begins with a line containing only the stanza name in square brackets. Each value is defined by a line that begins with the value name, followed by an equals sign, followed by the value itself.

Tool Stanzas

A stanza with the name of a GMA tool defines the style settings for that tool. Currently-supported tool names (i.e., the names of these stanzas) include just "`mapper`". The tool-name stanza may include the following definitions:

`fonts=`⟨***stanza***⟩ Specifies the stanza which defines the custom fonts to be used by this program.

`dierolls=`⟨***stanza***⟩ Specifies the stanza which defines the custom styling to be used for die roll displays.

Font Definitions

A font definition stanza defines named fonts with lines of the form ⟨*fontname*⟩=⟨*definition*⟩

The definition of the font is provided by giving option-value pairs. You need only provide the options which differ from the default for a given font for your system. If the value in any of these pairs contains spaces, it must be enclosed in curly braces. Options and values are separated from each other with whitespace.

The available options are:

`-family` The name of the font. The underlying Tk system provides the family names `Courier`, `Helvetica`, and `Times`,

which are representative names for a fixed-width, sans serif, and serifed font, respectively; these may not necessarily be fonts with those exact names. You may also use the name of any font known on your system. This name is case-insensitive.

`-size` The size in points of the font, if a positive value. If negative, its absolute value specifies the font size in pixels.

`-weight` One of `normal` or `bold`.

`-slant` One of `roman` or `italic`.

`-underline` A boolean value; if true, characters drawn in this font will be underlined.

`-overstrike` A boolean value; if true, characters drawn in this font will have a line struck through their centers. Note that there is a style setting (q.v.) which will specify overstriking for a specific display type. This is generally preferred to setting it font-wide so the same font may be used for values which are overstruck and for others which are not.

Example:

```
[mapper]
fonts=myfonts
[myfonts]
hack=-family Hack -size 12
fancy=-family {Fancy Script Regular} -size 24 -slant italic
```

Die Roll Style Stanzas

There are a number of die roll reporting styles in effect to display the various elements that make up a report of a die roll result. The color and font of each may be controlled by entries in this stanza. Any not provided in this stanza will assume their built-in default values.

`bg_`⟨***element***⟩`=`⟨***color***⟩ Sets the color drawn behind text for the die-roll element named.

`bg_list_even=`⟨***color***⟩ Sets the background color for even-numbered entries in the recent and preset die roll lists.

`bg_list_odd=`⟨***color***⟩ Sets the background color for odd-numbered entries in the recent and preset die roll lists.

`default_font=`⟨***name***⟩ — Use the user-defined font ⟨*name*⟩ for all element fonts where no explicit `font_`⟨*element*⟩ assignment exists in this file.

`fg_`⟨***element***⟩`=`⟨***color***⟩ — Sets the color of the text for the die-roll element named.

`fmt_`⟨***element***⟩`=`⟨***string***⟩ — Specifies the formatting to be used for the reported ⟨*element*⟩. By default, some descriptive punctuation, spacing, and text are presented around the variable values being reported. If these are not how you would like those elements to be shown, specify your own string in this configuration value. The ⟨*string*⟩ is a `printf`(3)-style format string, with a "`%s`" at the location you would like the data item(s) to appear.

Since whitespace is allowed around the keys and values in an INI file, which is automatically ignored, a mechanism is provided to allow your format string to explicitly contain leading and/or trailing spaces. If your format string (excluding any leading or trailing spaces) begins or ends with a vertical bar ("`|`"), the bar(s) mark the explicit end(s) of the string value. Thus, given the configuration file lines:

```
fmt_a=  (%s)
fmt_b=| (%s) |
fmt_c=| (%s)
fmt_d=  (%s) |
```

the resulting format string for element `a` will be "`(%s)`" (with all spaces stripped); the format for `b` will have a space before and after the `(%s)` part; `c` will have a space before its content; and `d` will have a space after its content.

`font_`⟨***element***⟩`=`⟨***name***⟩ — Sets the font used for the die-roll element named. This font name must be defined in the selected font definition stanza in this file.

`offset_`⟨***element***⟩`=`⟨***delta***⟩ — Display the text of this element ⟨*delta*⟩ pixels higher than the baseline of the surrounding text. The ⟨*delta*⟩ value may be negative to lower the text.

`overstrike_`⟨***element***⟩`=`⟨***bool***⟩ A boolean value; if true, text for this element will have a line struck through it.

`underline_`⟨***element***⟩`=`⟨***bool***⟩ A boolean value; if true, text for this element will be underlined.

The ⟨*element*⟩ names may be `best`, `bonus`, `constant`, `critlabel`, `critspec`, `dc`, `diebonus`, `diespec`, `discarded`, `exceeded`, `fail`, `fullmax`, `iteration`, `label`, `max`, `maximized`, `maxroll`, `met`, `min`, `moddelim`, `operator`, `repeat`, `result`, `roll`, `separator`, `short`, `subtitle`, `success`, `until`, or `worst`, the meanings of which are described in `gma-dice`(3). In addition, the elements `fullresult` and `title` control the display of the overall total and user-assigned title for the die roll, `comment` describes comments about the die roll from the server, `system` is for system-generated messages, and the elements `from`, `normal`, and `to` control the appearance of chat messages generally.

A ⟨*color*⟩ value may be the name of a color as known to the system (e.g., listed in the X11 ⟨*rgb.txt*⟩ file), such as "`alice blue`" or "`PaleGreen4`", or a hexadecimal color value in the format `#`⟨*rgb*⟩, `#`⟨*rrggbb*⟩, `#`⟨*rrrgggbbb*⟩, or `#`⟨*rrrrggggbbbb*⟩. System-defined names such as "`systemActiveAreaFill`" (macOS) and "`system3dLight`" (Windows) are also recognized where available. The X11 names are typically implemented by the underlying frameworks even on non-Unix-like systems so they are generally usable everywhere.

Example:

```
; This is an example style.conf file which
; shows how to customize the mapper tool.
[mapper]
dierolls=mydice
fonts=myfonts

[mydice]
fg_fullresult=red
font_fullresult=hack
fmt_roll=(rolled {%s})

[myfonts]
hack=\-family Hack -size 12 -weight bold
```

HISTORY

This feature appeared in version 3.33 of the mapper tool.

AUTHOR

Steve Willoughby / steve@madscience.zone.

SEE ALSO

`gma-dice`(3).

BUGS

The Tcl library support for reading INI files isn't as enhanced as the one in the Python library, so since the mapper uses this configuration file it needs to conform to the simpler rules supported by the Tcl library: no multi-line values, keys and values separated only by an equals sign, and only supports "`;`" as the comment character.

Since this is (eventually) intended to apply to more tools than the mapper, this really should have been located in `~/.gma` instead of `~/.gma/mapper`. A future version will either move the default location or search both directories.

gma.conf(5)

NAME

gma.conf – DEPRECATED GMA GM Console configuration file

DEPRECATED

This file is no longer used by GMA as of version 6.15. Instead, the built-in preferences editor should be used.

DESCRIPTION

The gma(6) program displays the main GM console with timekeeping controls, event trackers, initiative tracker, and so forth. There are a number of configuration options which control its behavior, all of which are read from the configuration file `~/.gma/gma.conf`.

This file conforms to the format of the standard so-called "Windows INI file". In a nutshell, it contains a number of *stanzas* which in turn contain a number of *options*.

A stanza begins with the stanza name in square brackets on a line by itself. Following this, each line defines an option in the format

⟨*name*⟩=⟨*value*⟩ or ⟨*name*⟩:⟨*value*⟩

There may be any amount of whitespace around the "=" or ":", which is ignored.

Long ⟨*value*⟩s may span multiple lines by indenting all lines after the first.

Lines beginning with ";" or "#" are comments and ignored.

GMA Stanza

The main stanza is named `[gma]` and controls how the rest of the stanzas in the file are read. It contains the following options:

`current_world=`⟨*name*⟩
: This selects which world stanza is in play. The stanza will be named `[world.`⟨*name*⟩`]` and contains the main game settings for GMA. This allows multiple different campaigns to be described in the `gma.conf` file. The one currently in play is selected by putting its name as the value for this option.

`dark=`⟨*bool*⟩
: This selects whether `gma` runs with a dark color theme. The ⟨*bool*⟩ value may be "`0`", "`false`", "`no`", or "`off`" to indicate that dark mode is not to be used, or "`1`", "`true`", "`yes`", or "`on`" to indicate that it should be on.

`network_profile=`⟨*name*⟩
: This selects which network stanza is in play. It will be named `[network.`⟨*name*⟩`]` and will contain the network settings to use for the current game. If the value of ⟨*name*⟩ is "`none`", then no network stanza will be used, and no connection will be attempted to a game server at all.

World Stanza

The campaign settings are contained in the world stanza, which has the name `[world.`⟨*name*⟩`]` where ⟨*name*⟩ is the value of the `current_world` option in the `[gma]` stanza.

The options defined here include:

`blur_hp=`⟨*pct*⟩
: The amount by which to approximate hit points as shown on creatures' health bars on the map. If 0, their actual hit point total will be shown. Otherwise, it indicates a percentage value such that hit points will be reported only in increments of that percentage of the total.

`calendar_type=`⟨*name*⟩
: The name calendar system in use, such as "`golarion`" or "`gregorian`".

`caster_levels=`⟨*list*⟩
: This option sets the caster level for spell-casting characters for the purposes of creating the menu button to select caster level when displaying spell descriptions. This is a multi-line value, with each caster described on a separate line of the value. Remember to indent all lines after the first one. For clarity, you may wish to just put "`caster_levels=`" on a line by itself, then put each caster's level data on subsequent indented lines after it.

  Each entry in this list has the following values, separated by vertical bars ("`|`"):

  ⟨***name***⟩ The name of the character as they appear on the map.

  ⟨***cl***⟩ Their caster level.

`db_name=`⟨*path*⟩
: The file pathname upon which the GMA database is based. This holds persistent game state data between invocations of the `gma` program. The actual pathname of the database file will be `~/.gma/`⟨*path*⟩`_base.db`.

`display_name=`⟨*title*⟩
: This controls the campaign name as displayed in the title bar of the program.

`initiative_backup_path=`⟨*path*⟩
: This specifies a pathname where the current initiative list will be

saved periodically throughout combat. In this way, `gma` can be restarted in case the game session had to stop mid-combat (or in case `gma` crashed). The backup file can be loaded back into `gma` to resume where you left off.

`initiative_seed=`⟨*list*⟩
: This option pre-seeds the initiative tracker with a set of combatants expected to be in every encounter, i.e., the PCs. This is a multi-line value, with each creature described on a separate line of the value. Remember to indent all lines after the first one. For clarity, you may wish to just put "`initiative_seed=`" on a line by itself, then put each creature's initiative data on subsequent indented lines after it.

  Each entry in this list has the following values, separated by vertical bars ("`|`"):

  ⟨***name***⟩ The name of the character as they appear on the map.

  ⟨***dex***⟩ Dexterity score

  ⟨***mods***⟩ Modifiers to initiative other than Dexterity.

  ⟨***con***⟩ Constitution score.

  ⟨***hp***⟩ Maximum hit points.

  ⟨***blur***⟩ The amount by which to approximate hit points as shown in the character's health bar on the map. If 0, their actual hit point total will be shown. Otherwise, it indicates a percentage value such that hit points will be reported only in increments of that percentage of the total. This overrides the global `blur_hp` option value for that character. This value is optional. If not present (i.e., only 5 values are given for this initiative_seed field), it defaults to the campaign setting for all creatures.

  ⟨***die_spec***⟩ The die-roll specification to use for rolling initiative for this character. This is optional. If not present (i.e., only 5 or 6 values are given for this initiative_seed field), it defaults to "`d20`". Note that this specification does not include any of the applicable modifiers to initiative in play (these will be added to the ⟨*die_spec*⟩ value automatically). However, this is useful if the character has a non-standard die roll for initiative. For example, if the character rolls two d20 dice and uses the higher roll for their initiative, then the ⟨*die_spec*⟩ would be set to "`d20 best of 2`".

`module=`⟨*path*⟩
: This causes `gma` to read the GMA Encounter (`.gmo`) file to be read and used with the initiative tracker. Without this option, the GM would need to manually open the encounter file using the application menu.

`password=`⟨*pass*⟩
: If using a game server, this supplies the GM password to use to authenticate the `gma` program.

Network Stanza

This stanza, which is named `[network.`⟨*name*⟩`]` where ⟨*name*⟩ is set in the value of the `network_profile` option of the `[gma]` stanza.

It has the following options:

`hostname=`⟨*host*⟩
: The hostname of the game server.

`port=`⟨*port*⟩
: The TCP port number of the game server.

EXAMPLE

```
[gma]
current_world=mycampaign
network_profile=mapservice

[world.mycampaign]
blur_hp=10
calendar_type=golarion
caster_levels=
    Bob|3
db_name=mycampaign
display_name=My Campaign
initiative_backup_path=/home/gm/games/backup.gmi
initiative_seed=
    Alice|16|4|14|20|0
    Bob|10|0|12|15|0
module=/home/gm/games/mycampaign.gmo
password=I@mTheGM!

[network.mapservice]
hostname=example.com
port=2323
```

SEE ALSO

`gma`(6).

AUTHOR

Steve Willoughby / steve@madscience.zone;

APPENDIX D

Game Program Reference

> I would like the world to remember me as the guy who really enjoyed playing games and sharing his knowledge and his fun pastimes with everybody else.
>
> —Gary Gygax

The following pages document the command-line (CLI) invocation and usage of the programs that comprise most of GMA. They are placed here in section 6 rather than the section 1 more commonly used for command-line tools because GMA is a set of game-playing programs, and section 6 is for documentation of game programs.

These manual entries are also available on the system where GMA is installed, in the form of online manual pages ("manpages"), which can be viewed using the `man` command of GMA (e.g., "`gma man [6] printchar`"). These pages are in section 6, so following the conventions for Unix manpages, their names have "(6)" appended to them.

gma-crxp(6)

NAME

gma crxp – Convert challenge rating to experience points for Pathfinder 1E

SYNOPSIS

```
gma crxp ⟨cr⟩ [...]
```

DESCRIPTION

For each challenge rating ⟨*cr*⟩ in the argument list, `crxp` prints the corresponding number of experience points a creature or hazard of that `CR` is worth.

`CR` values from 0–30 are supported, as well as `1/8`, `1/6`, `1/4`, `1/3`, and `1/2`. The latter are entered as shown (e.g., "`1/4`", not "`0.25`").

AUTHOR

Steve Willoughby / steve@madscience.zone.

gma-forumpost(6)

NAME

gma forumpost – prepare HTML forum posting for game session summaries

SYNOPSIS

```
gma forumpost [⟨inputfile⟩]
```

DESCRIPTION

This is a small helper script used for the author's game group but may be useful to others so is offered along with the rest of the tools just in case. Larry Wall was fond of saying that great software developers were gifted with virtues that included impatience, hubris, and laziness, and that these tended to drive more innovation and automation than others thought. This might be an example of that principle in action.

In our group, we use a typical online web forum to collaborate with each other. Part of that includes a space where we write up a summary of what happened during each game session. Being me, I like to do that with a certain amount of style, formatting it nicely with some included maps and other illustrations, HTML display elements that hide the blow-by-blow combat transcripts until you want to actually see them, etc.

After a few (dozen... score...) times of doing this, I got tired of typing all the HTML elements over and over, so I developed this script to do it for me, and reduced my effort to that of preparing a simpler script with some simple markup symbols instead of the HTML, including some that avoid collision with the HTML and JavaScript codes that are also in use.

Admittedly, some of these make more sense if you consider them as a quick short-hand way to get something into a file to be translated *into* something that makes sense instead of something that's supposed to make sense on its own.

Once the file is prepared, it is named on the command line or simply piped into this script. The HTML-rendered result is produced on its standard output.

The result is wrapped between `[html]` and `[/html]` tags so that it can be posted in our web forum, and have it include raw HTML codes. This requires the poster to have elevated privileges since posting raw, arbitrary HTML and JavaScript to a web forum is never something you would allow untrusted strangers to do to your forum. (In this case, the one doing this is not only our GM but also the system administrator and web forum maintainer, so that's already a given.)

MARKUP CODES

Generally, the file may contain any HTML and JavaScript elements desired.

Block Macros

The following major codes should be placed on a line by themselves. Anything else on the same line will be lost. They are always enclosed in double square brackets. Generally they introduce large content blocks or major mode changes of some sort.

`[[ID` ⟨*module-id*⟩ ⟨*game-id*⟩ ⟨*video-url-base*⟩`]]`
: This should be the first line of the file. It defines several ID numbers that are important for the entry:

  ⟨*module-id*⟩
  : The GM's identifier for the whole module (whether off-the-shelf or homebrew, we'll still call it a module). For example, for book 11 of our Age of Worms campaign, I used "`aw11`".

  ⟨*game-id*⟩
  : The session number for this module, starting with 1 and counting sequentially up from there.

  ⟨*video-url-base*⟩
  : The variable part of the YouTube URL where the video recording of our game was uploaded. This is the part after "`?v=`".

`[[COMBAT` ⟨*h*⟩`:`⟨*m*⟩`:`⟨*s*⟩`:`⟨*f*⟩ `[SURPRISE]]]` ... `[[/COMBAT]]`
: These tags enclose the play-by-play (or maybe more accurately, blow-by-blow) transcript of a combat encounter. The only required parameter is the timestamp in the video of where the combat starts, in hours, minutes, seconds, and frame number within the second. Optionally, a second parameter with the literal word SURPRISE may be given if the first round is a surprise round. This automatically includes a link to the video at the given timestamp, and places an inline image of the battle map assuming you have an image of it uploaded at the forum URL ⟨*image-url-base*⟩`/`⟨*module-id*⟩`/`⟨*module-id*⟩`.`⟨*game-id*⟩`.c`⟨*sequence*⟩`.png` where ⟨*image-url-base*⟩ comes from your GMA preferences and ⟨*sequence*⟩ is an automatically-incrementing sequence number starting at 0 for each `[[COMBAT]]` block encountered in this file. Note that `[[/COMBAT]]` marks the end of the combat block.

`[[ROUND` ⟨*number*⟩`]]`
: Inside a `[[COMBAT]]` block, use this to indicate the start of each new round after the first.

`[[DATETIME` ⟨*date*⟩`@`⟨*time*⟩`@`⟨*location*⟩ ⟨*weather*⟩ ⟨*degrees*⟩ `]]`
: Place a division in the summary that indicates a new date and time in the story, with the given ⟨*date*⟩ and ⟨*time*⟩ as written, and that the

story is now taking place at the given ⟨*location*⟩ in the game world. The ⟨*weather*⟩ value is one of the following keywords: `clear`, `cloudy`, `fog`, `rain`, `rain2`, `snow`, `storm`, or `dark`, to indicate that it is sunny (or clear if night time), overcast, foggy, raining, raining (with alternate symbol), snowing, stormy, or underground, respectively. If you put an asterisk at the end of the keyword (e.g., "`cloudy*`"), that means it's after dark as opposed to daytime. The temperature is in units of degrees Fahrenheit.

`[[TIME [`⟨*date*⟩`@]`⟨*time*⟩`]]`
: Like `[[DATETIME]]` but just indicates a new time during the same day (or—if the optional date and `@` sign are included—a new date and time) without all the other stuff.

`[[LOOT]]`
: Run `gma loot` to process the source file ⟨*module-id*⟩`.`⟨*game-id*⟩`.loot` with ⟨*module-id*⟩ in upper-case, inserting its HTML output at this place in the game summary.

In-line Shorthand Codes

The following short-hand markup codes may be used anywhere to facilitate easy and rapid entry of common things we use in game summary transcripts instead of the more tedious HTML codes we'd have to otherwise type directly.

`{`⟨*text*⟩`}`
: Place ⟨*text*⟩ in double quotation marks. This avoids typing `“` and `”` or getting the actual quotes confused with the JavaScript that was in the same file.

`{{`⟨*text*⟩`}}`
: Same as `{`⟨*text*⟩`}` except automatically boldfaces the quoted text. We use the convention of bolding spoken dialog so this just makes that convention easier without needing to explicitly add the bolface tags around the speech.

`{/`⟨*text*⟩`/}`
: Same as `{`⟨*text*⟩`}` but this indicates telepathic speech or internal thoughts so it is bolded and italicized but does not have quotation marks around it.

`'`⟨*text*⟩`'`
: Place ⟨*text*⟩ in single quotes.

`[|`⟨*name*⟩`|]`
: Set name in bold (usually used in combat to indicate the start of a person's turn).

`[(`⟨*name*⟩`)]`
: Set name in bold and parentheses (usually used in combat to indicate that the person took an out-of-turn action).

`[-`⟨*name*⟩`-]`
: Set name in bold and struck-through (usually used in combat to indicate thhat the person was unable to act this turn).

`[:`⟨*ooc-text*⟩`:]`
: Places ⟨*ooc-text*⟩ in steel blue color to mark it as "out-of-character" side commentary in the text.

`[/`⟨*text*⟩`/]`
: Places ⟨*text*⟩ in italics. Usually used for names of spells and magic items but can also be used for general emphasis.

`[*`⟨*text*⟩`*]`
: Marks ⟨*text*⟩ that is a direct excerpt from the published module for attribution purposes.

`\{`
: Renders a literal open curly brace.

`\}`
: Renders a literal close curly brace.

`\&`
: Renders a literal ampersand (i.e., literally sends `&`) in the output HTML.

`...`
: Renders ellipses as `…`

`---` Renders an em-dash.

`--` Renders an en-dash.

`\\`
: Renders a literal backslash.

`\.`
: Does nothing but can be used to cause other effects, e.g., `{\.{` would prevent two consecutive braces from being misinterpreted as a `{{` code.

`\$`
: Inserts a line break.

`$$`
: Inserts a paragraph break in a multi-paragraph run, i.e., inserts `</p><p>`.

FILES

The weather images need to be located at your ⟨*forum-image-url-base*⟩ directory (as indicated in your GMA preferences) with the names `weather/32/`⟨*weather-icon*⟩`.png` where ⟨*weather-icon*⟩ is, respectively, `day_clear`, `day_cloudy`, `day_fog`, `day_rain`, `day_rain2`, `day_snow`, `day_storm`, and `dark` for the daytime weather icons and `night_clear`, `night_cloudy`, `night_fog`, `night_rain`, `night_rain2`, `night_snow`, `night_storm`, and `dark` for nighttime weather.

Your combat images need to be uploaded to ⟨*forum-image-url-base*⟩`/`⟨*module-id*⟩`/`⟨*module-id*⟩`.`⟨*game-id*⟩`.c`⟨*sequence*⟩`.png`

A CSV format copy of your loot data shoud be uploaded to ⟨*forum-image-url-base*⟩`/`⟨*MODULE-ID*⟩`.`⟨*game-id*⟩`.loot.csv` where ⟨*MODULE-ID*⟩ is the ⟨*module-id*⟩ in upper-case letters.

BUGS

Currently only looks at the currently selected campaign world in the GMA preferences.

AUTHOR

Steve Willoughby / steve@madscience.zone.

gma-initiative-clock(6)

NAME

gma initiative-clock – GMA battle grid clock and initiative display

SYNOPSIS

gma `initiative-clock` [`-h`] [`-C` ⟨*path*⟩] [`-c` [`-d` ⟨*msec*⟩] ⟨*type*⟩] [`-f` ⟨*fontname*⟩] `-H` ⟨*hostname*⟩ [`-P` ⟨*pass*⟩] [`-p` ⟨*port*⟩] [`-u` ⟨*name*⟩]
gma `initiative-clock` [`--help`] [`--calendar` ⟨*type*⟩] [`--config` ⟨*file*⟩] [`--delay` ⟨*msec*⟩] [`--font` ⟨*fontname*⟩] `--host` ⟨*hostname*⟩ [`--password` ⟨*pass*⟩] [`--port` ⟨*port*⟩] [`--username` ⟨*name*⟩]

DESCRIPTION

The `initiative-clock` program is a stand-alone viewer for the GMA tool's game clock and initiative order display. This allows auxiliary viewers for this information during the game. It connects to the mapper service TCP port used by the `gma-mapper`(6) clients, passively listening to it for information to display.

OPTIONS

The following options control the behavior of `initiative-clock`.

`-C`, `--config` ⟨*path*⟩
: Read configuration values from the specified file. This is intended to allow you to use the mapper configuration file documented in `gma-mapper`(6) for this program as well, so it connects to the same server in the same way as the mapper does. Command-line parameters take precedence over the values in the configuration file. If a file called `~/.gma/mapper/mapper.conf` exists, it will be read by default unless this option is given to specify a different file to read. Only options recognized by `gma-mapper`(6) will be loaded from this file.

`-c`, `--calendar` ⟨*type*⟩
: Set the calendar type being used in the game. Possible values include `donuttus`, `golarion`, `gregorian`, or `tien`. The default is `golarion`.

`-d`, `--delay` ⟨*msec*⟩
: Set the delay between updates to the specified number of milliseconds. The default is 50, which is probably fine in most cases.

`-f`, `--font` ⟨*name*⟩
: Use the specified font ⟨*name*⟩ for the list of people in the initiative list. The default is "`Helvetica 24`".

`-H`, `--host` ⟨*name*⟩
: Connect to a map control service running on the designated host.

`-h`, `--help`
: Print the basic usage information for the program and exit.

`-P`, `--password` ⟨*pass*⟩
: If a password is required to connect to the server, use this option to provide it.

`-p`, `--port` ⟨*port*⟩
: Connect to the specified TCP ⟨*port*⟩ number on the host specified to the `-H` option. The default is port 2323.

`-u`, `--username` ⟨*name*⟩
: Log in to the server with the specified ⟨*name*⟩ rather than you logged-in system user account name.

SEE ALSO

`gma`(6), `gma-mapper`(5), `gma-mapper`(6).

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

This utility was first created by breaking out the initiative display code from the GM console, 5-Oct-2019.

gma-latex-macros(7)

See gma-modcleanup(6), p. 504.

gma-loot(6)

NAME

gma loot – Format post-game loot summary table report in HTML

SYNOPSIS

`gma loot` [`-cLs`] [`-o` ⟨*outputfile*⟩] ⟨*inputfile*⟩
`gma loot` [`--csv`] [`--no-links`] [`--strip`] [`--output` ⟨*outputfile*⟩] ⟨*inputfile*⟩

DESCRIPTION

This is a little script to make it easier to report on the loot obtained from a game session. Normally, in our games, we use a web forum to summarize the game play, with a summary of the loot included as a table at the bottom. However, this involved some tedious HTML table editing which this script seeks to eliminate.

OPTIONS

`--csv`
: (`-c`) Instead of the default HTML output format, generate a CSV output suitable for import into a spreadsheet. In this case the section names are folded into a column of their own and the order of the columns reordered slightly: ⟨*ID*⟩, ⟨*section*⟩, ⟨*quantity*⟩, ⟨*description*⟩, ⟨*notes*⟩, ⟨*caster level*⟩, ⟨*charges*⟩, ⟨*value*⟩, and ⟨*flags*⟩.

`--no-links`
: (`-L`) Don't render hyperlinks in the markup as links. Instead, render them as italics or underlined text as appropriate for their link type.

`--strip`
: (`-s`) Strip out the markup formatting codes from the description fields.

`--output`
: (`-o`) Write the output to the named ⟨*file*⟩. Specifying "-" as the filename causes `gma loot` to write to the standard output, which is the default behavior.

FILE FORMAT

Using this script, the GM now needs only prepare a simple text file describing the loot. This lists all the loose coinage collected on lines in the form

`$` ⟨*amount*⟩ ⟨*type*⟩

where ⟨*amount*⟩ is an integer number of coins and ⟨*type*⟩ is the coin type, typically `pp`, `gp`, `sp`, or `cp`. All of these will be collected together and the number of each kind of coin will be reported in a table.

The rest of the input file describes the items of loot picked up. It may be dividided into sections by starting a line with an = sign followed by a section title:

= ⟨*title*⟩

All other lines describe individual loot items. These are divided into eight fields, separated by vertical bars (`|`):

⟨*ID*⟩
: A unique identifier that the players and `GM` can use to refer back to this item later.

⟨*Flags*⟩
: A set of one-character flags which indicate special statuses for this item. they may appear in any order and may be any of the following:

  `+`
  : The players have detected a magic aura on the item.

  `$`
  : The item needs to be appraised.

  `?`
  : The item's properties are assumed to be identified because it resembles something the party is already familiar with.

  `a`
  : The item (typically armor of some sort) is designed for an animal rather than a humanoid.

  `c`
  : The item is cursed.

  `f`
  : The item is fungible as cash (it would be exchanged at full value on the market instead of sold at 1/2 retail price).

  `i`
  : The players have identified the item's properties.

  `m`
  : The item is masterwork. You may include this if the item also radiates magic, but in this case, only the magic flag will be displayed on output instead of the masterwork one.

  `t`
  : The price listed is the total for the whole collection, even though there is a listed quantity. This is used when there is a quantity of items, such as miscellaneous gemstones, but they may not all be of identical value (or for some other reason you don't want to list the individual unit price for them).

⟨*size*⟩
: One of the upper-case letters `D`, `F`, `T`, `S`, `M`, `L`, `H`, `G`, or `C` may be used as a flag to indicate that the item is a nonstandard size category, being designed for a creature that is diminutive, fine, tiny, small, medium, large, huge, gargantuan, or colossal, respectively.

⟨*Description*⟩
: The description of the item. The value in this field may be formatted using the GMA markup syntax.

⟨*Charges*⟩
: If not empty, this gives the integer number of charges remaining in the item.

⟨*Quantity*⟩
: If empty, it is assumed there is only one of this item. Otherwise, this gives the integer number of identical items of this kind that were found.

⟨*CL*⟩
: If not empty, this gives the caster level at which the item was created.

⟨*Value*⟩
: If not empty, this gives the floating-point value in gold pieces of the item (or of each item if there are multiple quantities of them).

⟨*Notes*⟩
: This field holds any miscellaneous notes you wish to make about the item.

Completely empty fields at the end of the line may be omitted entirely. Blank lines are ignored, as are lines which begin with octothorpes (`#`).

EXAMPLE

```
$30gp
$ 15 pp
$ 250cp
$ 100 gp
$ 12gp
=Gobin Lair
#ID|flags|description    |chg|qty|CL|gp|comments
  1|+    |//+1 dagger//
  2|     |crossbow
  3|     |bolts          |   | 16
=Guardroom
  4|+    |//+1 armor//   |   |  2
  5|+i   |//wand of CLW//| 13|  1| 1|  |found on shaman
```

BUGS

Compatibility Note

Starting with version 6.33, the meaning of the "`m`" flag changed from "magic item" to "masterwork item" and a new flag "+" was introduced to mean "magic item" from that point forward. These names make more mnemonic sense and in hindsight would have been chosen if we had known in advance there would have been a masterwork flag to begin with.

AUTHOR

Steve Willoughby / steve@madscience.zone.

gma-map-console(6)

NAME

gma map-console – GMA map service console (DEPRECATED)

SYNOPSIS

gma `map-console` [`-h`] [`-C` ⟨*path*⟩] `-H` ⟨*hostname*⟩ [`-P` ⟨*pass*⟩] [`-p` ⟨*port*⟩] [`-u` ⟨*username*⟩]
gma `map-console` [`--help`] [`--config` ⟨*path*⟩] `--host` ⟨*hostname*⟩ [`--password` ⟨*pass*⟩] [`--port` ⟨*port*⟩] [`--username` ⟨*username*⟩]

DESCRIPTION

This command is deprecated. The `map-console` program that is part of the Go-GMA project is the currently-supported one.

Normally, the GMA `gma-go-server`(6) orchestrates the communication between `gma-mapper`(6) clients and other clients such as `gma-initiative-clock`(6) and the `gma`(6) GM console. The `map-console` program provides a way to observe the communications going on and manually inject commands into the stream. It is intended only for basic debugging of `gma-go-server`(6), not for general use.

Obviously, this should be done with caution by someone intimately familiar with the protocol and the implications of injecting commands into the working system like this.

It connects to the mapper service TCP port used by the `gma-mapper`(6) clients, passively listening to it for information to display. It will silently respond to `MARCO` pings in the normal way expected for any client, printing only a blue dot on the screen each time it does so. Otherwise it prints each received command in a reasonable human-readable version.

At any point, a new command may be typed on the standard input. This command will be sent to the server as-typed (with minimal validation to ensure that it is at least able to be parsed as a TCL list).

OPTIONS

The following options control the behavior of `map-console`.

`-C`, `--config` ⟨*path*⟩
: Read configuration values from the specified file. This is intended to allow you to use the mapper configuration file documented in `gma-mapper`(6) for this program as well, so it connects to the same server in the same way as the mapper does. Command-line parameters take precedence over the values in the configuration file. If a file called `~/.gma/mapper/mapper.conf` exists, it will be read by default unless this option is given to specify

a different file to read. Only options recognized by `gma-mapper`(6) will be loaded from this file.

`-H`, `--host` ⟨*name*⟩
: Connect to a map control service running on the designated host.

`-h`, `--help`
: Print the basic usage information for the program and exit.

`-P`, `--password` ⟨*pass*⟩
: If a password is required to connect to the server, use this option to provide it.

`-p`, `--port` ⟨*port*⟩
: Connect to the specified TCP ⟨*port*⟩ number on the host specified to the `-H` option. The default is port 2323.

`-u`, `--username` ⟨*name*⟩
: Log in to the server with the specified ⟨*name*⟩ rather than you logged-in system user account name.

SEE ALSO

`gma`(6), `gma-mapper`(5), `gma-mapper`(6), `gma-go-server`(6).
There is also a different implementation of `map-console`(6) in the Go-GMA package. See p. 581 for details on that program.

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

This utility was first created 21-Feb-2020.

BUGS

This program blindly assumes that the user's terminal understands ANSI standard escape codes to produce colored text.

gma-map-lock(6)

NAME

gma map–lock – Lock or unlock map objects

SYNOPSIS

```
gma map-lock [-huv] ⟨map-files⟩...
gma map-lock [--help] [--unlock] [--verbose] ⟨map-files⟩...
```

DESCRIPTION

This utility reads in one or more map files for the `gma-mapper`(6) program. After renaming each input file to a file with a `.bak` suffix, it writes a new version back to the original file name but with all lockable map objects locked. option is given, no lasting changes will be made.

OPTIONS

The following options control the behavior of `map-lock`.

`-h`, `--help`
: Print usage information and exit.

`-v`, `--verbose`
: Print verbose information about the program's operation.

`-u`, `--unlock`
: Rather than locking all map objects, unlock them instead.

SEE ALSO

`gma-mapper`(5), `gma-mapper`(6).

AUTHOR

Steve Willoughby / steve@madscience.zone

gma-map-reduce(6)

NAME

gma map–reduce – Reduce map files by removing clutter

SYNOPSIS

```
gma map-reduce [-hmv] ⟨map-files⟩...
gma map-reduce [--help] [--merge] [--verbose] ⟨map-files⟩...
```

DESCRIPTION

This utility reads in one or more map files for the `gma-mapper`(6) program. After renaming each input file to a file with a `.bak` suffix, it writes a new version back to the original file name but with duplicate objects removed. Objects with the same name or ID (depending on type) that occur later in the file will be kept while the earlier ones will be removed.

This is particularly useful if you run `gma-rendersizes`(6) multiple times for the same images (say, to add more sizes or types of image for the same token), causing multiple definitions to pile up in the map file.

OPTIONS

The following options control the behavior of `map-reduce`.

`-h`, `--help`
: Print usage information and exit.

`-v`, `--verbose`
: Print verbose information about the program's operation.

`-m`, `--merge`
: Image definitions with the same name are merged rather than simply replaced. This means that if you define a few sizes of an image and then later define a different set of sizes for the same image, all of the sizes will be retained. Without this option, only the last definition in the file for the given image will be kept.

SEE ALSO

`gma-mapper`(5), `gma-mapper`(6), `gma-rendersizes`(6).

HISTORY

`map-reduce` first appeared in version 5.6 of GMA Core.

AUTHOR

Steve Willoughby / steve@madscience.zone

BUGS

Apologies to fuctional programming fans for the name of this script.

gma-map-translate(6)

NAME

gma map–translate – Move all objects in the map files by a given vector

SYNOPSIS

gma `map-translate` [`-hlv`] [`-i` ⟨*obj_id*⟩ | /⟨*obj_id_regex*⟩] [`-I` ⟨*img_name*⟩ | /⟨*img_name_regex*⟩] [`-x` ⟨*dx*⟩ | `@`⟨*x*⟩] [`-y` ⟨*dy*⟩ | `@`⟨*y*⟩] [`-z` ⟨*dz*⟩ | `front` | `back` | `@`⟨*z*⟩] ⟨*map-files*⟩...
gma `map-translate` [`--help`] [`--id` ⟨*obj_id*⟩ | /⟨*obj_id_regex*⟩] [`--image-name` ⟨*img_name*⟩ | /⟨*img_name_regex*⟩] [`--list`] [`--verbose`] [`-x` ⟨*dx*⟩ | `@`⟨*x*⟩] [`-y` ⟨*dy*⟩ | `@`⟨*y*⟩] [`-z` ⟨*dz*⟩ | `front` | `back` | `@`⟨*z*⟩] ⟨*map-files*⟩...

DESCRIPTION

This utility reads in one or more map files for the `gma-mapper`(6) program. After renaming each input file to a file with a `.bak` suffix, it writes a new version back to the original file name but with all map elements changed so their reference points are (x + ⟨*dx*⟩, y + ⟨*dy*⟩).

Additionally, objects with `points` attributes have those points translated by (+⟨*dx*⟩, +⟨*dy*⟩).

If the `-z` option is given, the objects' stacking order is adjusted.

OPTIONS

The following options control the behavior of `map-translate`.

`-h`, `--help`
: Print usage information and exit.

`-i`, `--id` ⟨*obj_id*⟩
: Only change the object with the specified ⟨*obj_id*⟩. If ⟨*obj_id*⟩ begins with a slash, the remainder is a regular expression. All objects whose ID matches this pattern are affected.

`-I`, `--image-name` ⟨*name*⟩
: Only change images with the specified ⟨*name*⟩. If ⟨*name*⟩ begins with a slash, the remainder is a regular expression. All images whose name matches this pattern are affected.

`-l`, `--list`
: List the objects to standard output which are affected by the operation. If no `-x`, `-y`, or `-z` option is also given, no changes will be made.

`-v`, `--verbose`
: Print verbose information about the program's operation.

`-x` ⟨*dx*⟩ | `@`⟨*x*⟩
: Add ⟨*dx*⟩ to every object's *x* coordinate. If this value begins with an at-sign ("`@`"), the coordinate is set to the specified value rather than being translated by ⟨*dx*⟩.

`-y` ⟨*dy*⟩ | `@`⟨*y*⟩
: Add ⟨*dy*⟩ to every object's *y* coordinate. This takes the same forms as the `-x` option.

`-z` ⟨*dz*⟩ | `@`⟨*z*⟩ | `front` | `back`
: As with `-x` and `-y`, but this adjusts the *z* coordinate (the object stacking order on the canvas, with higher values drawn on top of lower-numbered ones). The value `back` forces the object to be at the bottom of the set of objects in the file, while the value `front` ensures it will be on top. This sense of "front" or "back" includes all objects in all named files. (Of course, there's no guarantee it will be on top of objects loaded from other files not included in this execution of `map-translate`.)

  Note that if you translate the room locations on a map for which you have already set up encounters, you will need to adjust the `spawn` fields for creatures so they appear on the map in the new locations.

SEE ALSO

`gma-mapper`(5), `gma-mapper`(6).

AUTHOR

Steve Willoughby / steve@madscience.zone

gma-mapper(6)

NAME

gma 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⟩] [-D] [-d] [-f ⟨fmt⟩] [-G
⟨n⟩[+⟨x⟩[:⟨y⟩]]] [-g ⟨n⟩[+⟨x⟩[:⟨y⟩]]] [-h ⟨hostname⟩] [-k] [-L] [-l] [-M
⟨module⟩] [-n] [-P ⟨pass⟩] [-p ⟨port⟩] [-S ⟨profile⟩] [-t ⟨transcript⟩]
[-u ⟨name⟩] [-X ⟨proxyhost⟩] [-x ⟨proxyurl⟩] [--button-size
small|medium|large] [--chat-history ⟨n⟩] [--curl-path ⟨path⟩]
[--curl-url-base ⟨url⟩] [--dark] [--debug-protocol]
[--mkdir-path ⟨path⟩] [--nc-path ⟨path⟩] [--no-animate]
[--no-blur-all] [--preferences ⟨path⟩] [--scp-dest ⟨dir⟩]
[--scp-path ⟨path⟩] [--scp-server ⟨hostname⟩] [--ssh-path
⟨path⟩] [--update-url ⟨url⟩]

gma mapper [-display ⟨name⟩] [-geometry ⟨value⟩] [⟨other wish
options⟩] -- [--[no-]animate] [--[no-]blur-all] [--blur-hp ⟨pct⟩]
[--button-size small|medium|large] [--config ⟨file⟩]
[--chat-history ⟨n⟩] [--curl-path ⟨path⟩] [--curl-url-base ⟨url⟩]
[--dark] [--debug] [--debug-protocol] [--guide ⟨n⟩[+⟨x⟩[:⟨y⟩]]]
[--host ⟨hostname⟩] [--image-format ⟨fmt⟩] [--keep-tools]
[--list-profiles] [--major ⟨n⟩[+⟨x⟩[:⟨y⟩]]] [--mkdir-path ⟨path⟩]
[--module ⟨module⟩] [--nc-path ⟨path⟩] [--no-chat] [--password
⟨pass⟩] [--port ⟨port⟩] [--preferences ⟨path⟩] [--preload]
[--proxy-host ⟨proxyhost⟩] [--proxy-url ⟨proxyurl⟩] [--scp-dest
⟨path⟩] [--scp-path ⟨path⟩] [--scp-server ⟨hostname⟩] [--select
⟨profile⟩] [--ssh-path ⟨path⟩] [--transcript ⟨filename⟩]
[--update-url ⟨url⟩] [--username ⟨name⟩]
```

Options cannot be combined into a single CLI argument. They are evaluated in the order they are received.

As shown above, if you have the GMA Core package installed, you may invoke the `mapper` program via the main `gma` program as

```
$ gma mapper ...
```

If you installed `mapper` as a stand-alone package by itself, you need to invoke it by running the `mapper.tcl` program found in the `bin` directory

of the `GMA-Mapper` package as unpacked or cloned from git. Depending on your system's particular configuration, you may be able to run it directly, as

```
$ bin/mapper.tcl
```

or you may need to explicitly run the `wish`(1) interpreter in a form such as

```
$ wish bin/mapper.tcl
$ wish8.6 bin/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.

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

CONFIGURATION

Starting with version 4.4, the mapper now uses an integrated preferences editor for its configuration. This eliminates the necessity of controlling the mapper via command-line options and/or configuration files, although those may still also be used.

Selecting Edit » Preferences... from the menu calls up the preference editor window. There you can enter global preference settings as well as server-specific ones. In the latter case, multiple server profiles may be created. Selecting one of these profiles in the list will make it the current profile. Every time the mapper is started, that server profile will be loaded and a connection opened to the server named in it.

If you wish to simply select a different server profile as the current one and connect to it, access the Play » Connect to menu and choose the desired server profile.

Some preferences may not take effect until the mapper is restarted. When leaving the editor window, you will be given the opportunity to restart the mapper if you desire.

Once the editor window is used, a preferences data file will be created as `~/.gma/mapper/preferences.json`. If that file exists, it will always be loaded when the mapper starts. In this case, the mapper will not load the

`~/.gma/mapper/mapper.conf` file anymore unless explicitly requested via command-line option.

If any command-line options are given, they will override the settings loaded at program start-up from the `preferences.json` file.

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, --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.)

`-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.

`-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 an octothorpe ("#") 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.

`--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`, `--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.

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

`--debug-protocol`
: Turns on debugging messages showing all interactions between client and server, and the client-side procesesing that is done to support that interaction.

`-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 below).

`--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.

`-f`, `--image-format gif|png`
: Sets the preferred image file format to request when retrieving images from the server to display on the map.

`-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`, `--list-profiles`
: List all the available profiles in the current preferences settings (either the default set for the mapper client or the one specified with the `--preferences` option). This list is printed to the standard output and then the mapper exits. These are the profile names which may be given to the `--select` option.

`-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.

`--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.

`--preferences` ⟨*path*⟩
: Use the specified ⟨*path*⟩ instead of `~/.gma/mapper/preferences.json` for the set of user preferences to use for this invocation. This allows a

completely separate set of preferences to be used for different users or purposes. This command may not appear in a configuration file (and is ignored if it is). It must be given only on the command line.

`-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.

`-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`.)

`--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).

`-S`, `--select` ⟨*profile*⟩
: Selects the named server ⟨*profile*⟩ from the `preferences.json` file without making it the selected profile for other invocations of the mapper.

`--ssh-path` ⟨*path*⟩
: Specify the path to the `ssh`(1) program used to send files *to* the storage server (authorized users only).

`-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.

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.

Delete (aka 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 `gma-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 ($\langle t \rangle$–$\langle l \rangle$<=–$\langle x \rangle$), 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 "$\langle hp \rangle$/$\langle max \rangle$" where $\langle hp \rangle$ is the current number of hit points remaining, out of a maximum of $\langle max \rangle$ hit points. If non-lethal damage has been suffered, then the display is "$\langle hp \rangle$($\langle nl \rangle$)" where $\langle nl \rangle$ 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.

To the right of the entry line there is also a checkbox with the letter "M". If selected, this allows GMA markup formatting codes to be typed in your messages. This allows, for example, boldface text to be specified by surrounding text in ****double asterisks**** and italics by surrounding text in //*double slashes*//. See `gma-markup`(7) (p. 690) for a more complete explanation.

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 `gma-dice-syntax`(7). 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 `gma-dice`(5).

If your (or the system-wide) set of presets includes variables, modifiers, or lookup tables, they will appear in this pane as well.

Display Initiative Clock This button calls up the combat initiative window which displays a set of timers which keep track of interesting events such as how long spells or other timed effects have left until they expire, the running campaign time clock, initiaitive turns, and other tactical information of use during combat.

Request a New Timer Clicking on this button allows you to make a request to the GM to add a new timer to the initiative clock display for something you want to track in the game.

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.

INSTALLATION

To run the mapper, you'll need an up-to-date Tcl/Tk interpreter (8.6 or later), and the tcllib and tklib standard libraries.

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), `gma-dice`(3), `gma-dice`(5), `gma-mapper`(5), `gma`(6), `gma-dice-syntax`(7), `gma-mapper-protocol`(7).

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.

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. See the detailed installation and usage tutorials in the *GMA Game Master's Guide*.

FILES

`~/.gma/mapper/preferences.json`
: Preferences data storage file, normally edited via the in-application preferences editor window.

`~/.gma/mapper/mapper.conf`
: (DEPRECATED) Default configuration file read if no explicit `-C` or `--config` option is given and no `preferences.json` file was read.

`~/.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.

Some of the menu behavior required by the mapper client has been shown not to work with Tcl/Tk version 8.6, so the mapper avoids using those features when the Tcl/Tk version is less than 8.7.

DEPRECATED FEATURES

The preferred way to configure `mapper` is to use the built-in preferences editor. This obsoletes the use of the older simple configuration file which needed to be edited by hand, and makes it unnecessary to use CLI options to specify the runtime parameters for the mapper's operation. The use of the legacy configuration file is still supported, however, as is the use of CLI options where desired for *ad-hoc* changes from the saved preferences.

However, the switch to the preferences system makes the following CLI options now obsolete and they no longer perform any function: `--generate-config`, `--generate-style-config`, `-s`, and `--style`.

The `-c` (`--character`) option is no longer supported.

gma-mapservice(6)

NAME

gma mapservice – GMA battle grid map server (DEPRECATED)

SYNOPSIS

gma `mapservice` [`-h`] [`-d` ⟨*path*⟩] [`-I` ⟨*mins*⟩] [`-i` ⟨*init-file*⟩] [`-l` ⟨*log-file*⟩] [`-P` ⟨*pass*⟩] [`-p` ⟨*port*⟩] [`-S` ⟨*state-file*⟩] [`-s` ⟨*pass-file*⟩]
gma `mapservice` [`--help`] [`--dice-preset` ⟨*path*⟩] [`--init-file` ⟨*init-file*⟩] [`--log-file` ⟨*log-file*⟩] [`--password` ⟨*pass*⟩] [`--password-file` ⟨*pass-file*⟩] [`--port` ⟨*port*⟩] [`--save-interval` ⟨*mins*⟩] [`--state` ⟨*state-file*⟩]

DESCRIPTION

This command is **deprecated**. The current implementation of the map service is now the **gma-go-server**(6) program in the Go-GMA package.

The individual `gma-mapper`(6) clients used by players in a game may keep in contact with one another so that they all display the same contents. A change made on one client (moving a creature token or adding a room, say) appears on all the others. This is accomplished by starting a `MapService` process and having all of the `mapper` clients connect to it via their `--host` and `--port` options.

Once connected, the `MapService` will send an initial greeting that may define a list of player character tokens to appear on the `mapper` context menus, or any other useful information the clients need to have at startup time. It may, at the GM's option, even initialize the client to show the full current game state.

From that point forward, the `MapService` is a simple echo relay between the clients, so they communicate with each other via the service. The server also tracks the commands it sees, so that it maintains a notion of the current state of the game. Clients may re-sync with the server in case they restart or otherwise miss any updates so they match the server's state. The server may respond directly to some client queries (e.g., "`AI?`") if it knows the answer rather than referring the query to the other clients.

To guard against nuisance or malicious port scans and other superfluous connections which don't proceed with the normal activity, the server will automatically drop any clients which don't authenticate within a few poll intervals. (In actual production use, we have observed some automated agents which connected and then sat idle for hours, consuming server resources.)

OPTIONS

The following options control the behavior of `MapService`.

`-h`, `--help`
: Print a usage summary and exit.

`-d`, `--dice-preset` ⟨*path*⟩
: Enable server-side saving of user die-roll presets. These will be saved in a file at the specified ⟨*path*⟩.

`-i`, `--init-file` ⟨*init-file*⟩
: Each line in ⟨*init-file*⟩ will be sent verbatim to each client upon connection to the server. The lines of this file must therefore be valid mapper protocol commands as documented in `gma-mapper`(6). The following special commands may appear in this file. These are not sent to the clients directly, but trigger other setup events in the server itself:

  `SYNC`
  : Enables sync-on-connect mode. With this set, the server will automatically perform the effects of a `SYNC` command for each client as it connects (after authentication if that is required).

  `LOAD` ⟨*filename*⟩
  : The ⟨*filename*⟩ (which is a pathname local to the server) is read, expecting it to be in the format documented in `gma-mapper`(5). Any image definition statements found (`I` records) are remembered so that the server can respond to subsequent `AI?` requests about them.

`-l`, `--log-file` ⟨*log-file*⟩
: Append a record of server actions to the specified file. This is not a complete record. Currently only connection information is recorded, although that may change in the future.

`-P`, `--password` ⟨*pass*⟩
: The server will require clients to authenticate with the specified password (⟨*pass*⟩). Without this parameter (or `--password-file`), the server allows any client to connect and participate without any authentication.

`-s`, `--password-file` ⟨*pass-file*⟩
: Same as `--password`, but the password is read from the specified file rather than from the command line. The first line from the file is read to obtain this password. The second line is read as the `GM`-level password.

  Subsequent lines, if any, give specific passwords for individual users. These have the form ⟨*username*⟩`:`⟨*password*⟩ Any login for the given

⟨*username*⟩ must use the individual password for that user to successfully authenticate to the server. Any ⟨*username*⟩ not listed in the password file (other than "gm") will successfully authenticate if the general-use shared password (the first line of the password file) is given.

`-p, --port` ⟨*port*⟩
: The service will accept incoming connections on the specified TCP port. The default is 2323.

`-I, --save-interval` ⟨*mins*⟩
: If the `--state` option is given, this sets the frequency (in minues) at which the game state is saved to ⟨*state-file*⟩. The default is every 10 minutes.

`-S, --state` ⟨*state-file*⟩
: Load the initial game state from ⟨*state-file*⟩ if it exists. Periodically save the game state back to this file while the server is running.

SECURITY

The authentication system employed here is simplisitic and not ideal for general use, but is considered to be good enough for our purposes here, since the stakes are so low :) It is intended just to discourage cheating at the game by looking at spoilers or direct messages intended for other users, not for any more rigorous protection.

The main weakness of the system is that passwords are stored in plaintext on the server, which means it is critical to secure the password file and the system itself. Caution your players to use a password for the mapper that is different from any other passwords they use (which should be the password practice people observe anyway). A breach that reveals passwords from the server's file would then only allow an imposter to log in to your map service, which admittedly is more of an inconvenience than a serious security issue, assuming you use your map server just for playing a game and not for the communication of any sensitive information.

Don't use the GMA mapper server for the communication of sensitive information. It's part of a game. Just play a game with it.

SIGNALS

The map service responds to the following signals while running. These actions may not be taken immediately but should happen within a few seconds.

`HUP`
: This signal causes the server to unconditionally save its current state.

`INT`
: This signal causes the server to save state if needed and shut down gracefully.

`USR1`
: This signal causes the server to dump a human-readable description of the current game state on its standard output.

SEE ALSO

`gma`(6), `gma-mapper`(5), `gma-mapper`(6), `gma-go-server`(6).

AUTHOR

Steve Willoughby / steve@madscience.zone.

BUGS

The code which drops unauthenticated connections doesn't correctly consider if the server is using authentication at all, so until that is fixed, it is effectively required for the server to use passwords.

gma-markup(6)

NAME

gma markup – Render GMA markup text into plain text, PostScript, or HTML

SYNOPSIS

```
gma markup -h
gma markup --help
gma markup [-Hps] < ⟨input⟩ > ⟨output⟩
gma markup [--html] [--ps] [--syntax] < ⟨input⟩ > ⟨output⟩
```

DESCRIPTION

GMA uses a simple markup syntax described in `gma-markup`(5) for free-form text content. The `gma markup` program documented here is a standalone utility to render text in this format into various output forms.

In normal operation, `markup` acts as a filter, converting the data sent to its standard input to its standard output.

Note that there is a version of this utility that appears in both the Python-based GMA Core (invoked as `gma markup`) and in the Go-GMA project (invoked as `gma go markup`). They serve the same function, but each uses the GMA markup implementation in their respective languages.

OPTIONS

The following options control the behavior of this program.

`-h`,`--help`
: Print a summary of the command-line usage and exit.

`-H`,`--html`
: Render the output as HTML instead of the default plain text.

`-p`,`--ps`
: Render the output as PostScript instead of the default plain text. This isn't sufficient for use on a PostScript device without some up-front initialization and inclusion of GMA's usual PostScript preamble, but where free-form markup-formatted text would appear in the PostScript document generated by GMA, this is the content that would be used there.

`-s`,`--syntax`
: Instead of reading standard input, render a description of the markup syntax as expected by this program.

SEE ALSO

`gma`(6), `gma-markup`(7), `gma-preamble`(7).

AUTHOR

Steve Willoughby / steve@madscience.zone.

gma-mkmap(6)

NAME

gma mkmap – generate PostScript dungeon maps from description files

SYNOPSIS

`gma mkmap` [`-adDEgr`] [`-x` ⟨*grids*⟩] [`-y` ⟨*grids*⟩] [⟨*descfile*⟩]

DESCRIPTION

`Mkmap` reads in the named file (or standard input) containing a description of a dungeon room, and writes to standard output a PostScript image of that room, drawn with 5' squares. The output image is rendered to the 2"=10' scale that seems to work well with miniatures.

If the image takes up more than one page, `mkmap` will split it up as necessary, and place coordinates at the top of the pages to indicate how the pages should be pasted together after printing.

OPTIONS

The following options may appear before ⟨*descfile*⟩ on the command line:

`-a`
: Automatically scale the page size to the number of rows and columns of characters in the "sketch" portion of the data file. This will not be less than the values given for `-x` and `-y` options (or their defaults). It includes non-printing characters in the sketch, so you could have a page size larger than the printed content.

`-D`
: Assume 24×36" D-size plot paper.

`-d`
: Add some debugging information to the output. This will be inserted as PostScript comments.

`-E`
: Assume 36×48" E-size plot paper.

`-g`
: Add gridlines under the map image every 25' (5 squares).

`-r`
: Rotate image 90 degrees on the paper (landscape).

`-x` ⟨*grids*⟩
: Set the horizontal page size to the specified number of 1" grids (5 scale feet per grid). The default is 8.

`-y` ⟨*grids*⟩
: Set the vertical page size to the specified number of 1" grids (5 scale feet per grid). The default is 11.

DESCRIPTION FILE FORMAT

The first line of a description file is the room ID, which will be printed at the top of each page (followed by the alignment coordinates).

Following that is a "rough sketch" of the room's area. This is done by arranging a grid of "`X`" characters in the file, where each `X` represents a 5×5 foot square of the room's floorplan.

This is the bare minimum for a room description, and will produce a scale gridmap of the room to match the pattern `X`'s given. More detail may be added by placing the following commands after the `X` grid in the file.

Other characters in this portion of the file won't be interpreted as grids, but will influence the `-a` option's calculation of page size.

Detail Commands

In the descriptions that follow:

`[`⟨*x*⟩`,`⟨*y*⟩`]`
: refers to a coordinate pair (⟨*x*⟩,⟨*y*⟩) where the specified detail is to begin. The square brackets are part of the syntax here. If ⟨*x*⟩ and/or ⟨*y*⟩ are preceeded by a minus sign, then that value is taken to be a number of 5' squares rather than feet.

⟨*o*⟩
: refers to an orientation for the effect. Orientations include two letters: `t` (top), `b` (bottom), or `c` (center), followed immediately by one of: `l` (left), `r` (right), or `c` (center).

All units are in scale feet unless otherwise noted. Following this common preamble is the actual command describing the detail desired. These are described below:

`[`⟨*x*⟩`,`⟨*y*⟩`]` ⟨*o*⟩ `arc` ⟨*r*⟩ ⟨*a1*⟩ ⟨*a2*⟩ `[`⟨*sx*⟩`,`⟨*sy*⟩`]` `c=`⟨*color*⟩ `t=`⟨*thickness*⟩
: Draw an arc from (⟨*x*⟩,⟨*y*⟩) for radius ⟨*r*⟩ starting at angle ⟨*a1*⟩ and going to angle ⟨*a2*⟩. Odd aspect ratios may be specified (if you don't want a circle) by giving scale values for ⟨*sx*⟩ and ⟨*sy*⟩ parameters (default is `[1,1]`). Color may be specified (default is 1), as may thickness (in points) of the lines (default is 4pt). The line around the arc is always black. The area inside is colored as indicated by the `c=` parameter. This value is either a real-number grayscale value from 1

(white) to 0 (black) or three real number values ⟨*r*⟩/⟨*g*⟩/⟨*b*⟩ for red, green and blue levels (with slashes separating them) where `1/1/1` is white, `0/0/0` is black, and `0.482/0.408/0.933` is a medium slate blue. If the color value is negative, e.g., `c=-1`, then the arc will not be filled with a color.

`[`⟨*x*⟩`,`⟨*y*⟩`]` ⟨*o*⟩ `box` ⟨*w*⟩`x`⟨*h*⟩ `c=`⟨*color*⟩ `t=`⟨*thickness*⟩
Draw a box (⟨*x*⟩,⟨*y*⟩) feet from room origin (upper left), with width ⟨*w*⟩ and height ⟨*h*⟩. Color and thickness as for `arc` command. Orientation is ignored. If the thickness is zero, then no line will be drawn around the box. Otherwise, the line around the box is always black. The area inside the box is filled in with the color specified in the `c=` option, or no fill if c<0.

`[`⟨*x*⟩`,`⟨*y*⟩`]` ⟨*o*⟩ `path` ⟨*w*⟩`x`⟨*h*⟩ `c=`⟨*color*⟩ `t=`⟨*thickness*⟩ ⟨*list...*⟩
Draw a line through the given path starting at (⟨*x*⟩,⟨*y*⟩) and turning in units of ⟨*w*⟩ wide by ⟨*h*⟩ high. The list of directions to turn are space separated and may contain the following. They may also be grouped into sets (comma-separated) to achieve various slopes of diagonal lines. `u` (up), `d` (down), `l` (left), and `r` (right).

`[`⟨*x*⟩`,`⟨*y*⟩`]` ⟨*o*⟩ `door` ⟨*o2*⟩
Place a door symbol in a wall. ⟨*x*⟩ and ⟨*y*⟩ are in units of grid-squares (i.e., 5'/unit). So don't prefix them with minus signs, unless you intend for them to be in units of 25' squares each. The ⟨*o*⟩ parameter indicates where on that grid-square's border the door should be drawn. For corners, the ⟨*o2*⟩ parameter may be `h` or `v` and specifies whether the door may be oriented horizontally or vertically. You may append an `x` at the end of ⟨*o2*⟩ to indicate that the door should be pushed down or right 1/2 square, so that double doors may be constructed.

`[`⟨*x*⟩`,`⟨*y*⟩`]` ⟨*o*⟩ `ps` ⟨*string*⟩
Insert arbitrary PostScript commands for detail work. The command string will be interpreted in an environment scaled to "scale feet" (14.4pt per scale foot), with the (0,0) coordinate origin at the (⟨*x*⟩,⟨*y*⟩) point specified by the command (and as usual with this file format, the ⟨*y*⟩ axis increases ⟨*down*⟩ from the origin).

EXAMPLE

```
CAVE1
...X....
...X....
..XXX...
XXXXXXXX
XXXXXXXX
```

```
XXXXXXXX
XXXX.XXX
[15,0] cc path 5x5 d d d,l l d,l d d d r u,r d,r r u u,r u r d
d, l d r r r, u u u u,l u,l l l u u
```

HISTORY

Originally created ca. 1994–5; enhanced Jan 2003 to include page scaling.

AUTHOR

Steve Willoughby

BUGS

This is a pretty simple-minded Perl script, and would no doubt be easy to confuse with weird input data.

gma-modcleanup(6)

NAME

gma modcleanup – prepare LaTeX module text after importing from other media

SYNOPSIS

```
gma modcleanup
```

DESCRIPTION

This is a small helper script used for the author's game group but may be useful to others so is offered along with the rest of the tools just in case.

I like to put together my local version of the modules I run using LaTeX which allows me to make all the custom modifications I need to make for my local group and still have a nicely typeset book to work with that is pleasant for me to use and easy to find the information I need. (And I just like working with LaTeX that much that I like having an excuse to use it for things.)

However, when copying and pasting various and sundry things from other sources like PDF files or websites to include into my local copy of a module, that may include weird characters that make editing the source file challenging. Even for original text, there are some common conventions that this script converts to macros that my common document style definition uses in order to facilitate easy and consistent module typesetting.

This script acts as a filter, translating the data on its standard input to its standard output. It is typically invoked from inside my text editor (vim) with a command like "`:'a,.!gma modcleanup`"

TRANSFORMATIONS

The transformations this script makes includes the following.

⟨*digits*⟩ `ar` ⇒ ⟨*digits*⟩`~\ar{}`
: Typesets dates in Absalom Reckoning.

`47`⟨*xx*⟩ ⇒ `\xxx{47`⟨*xx*⟩`}`
: Marks any recent dates for followup in case I need to adjust them for our campaign, unless they've already been marked (by having a brace immediately before the first digit).

`area` ⟨*id*⟩ ⇒ `\area{`⟨*id*⟩`}`
: Converts things that look like references to dungeon area numbers to use the macros designed to properly typeset them.

`areas` ⟨*id1*⟩...⟨*id2*⟩ ⇒ `\areas{`⟨*id1*⟩`}{...}{`⟨*id2*⟩`}`
: As above. Recognizes an en-dash between ⟨*id1*⟩ and ⟨*id2*⟩.

`page` ⟨*n*⟩ ⇒ `page~\pageref{orig-page-`⟨*n*⟩`}`
: A reference in whatever original document to a page number is changed to a generic reference here that we can define when we see where in the text it actually refers to. So if the generated reference is something like `\pageref{orig-page-12}` then at the very least at the destination text place a `\label{orig-page-12}`, but a better approach would be to replace this with a more meaningful reference like `\pageref{goblin-king}` and define `\label{goblin-king}` at the point where that is pointing to.

`-`⟨*digits*⟩ ⇒ `$-$`⟨*digits*⟩
: Convert Unicode minus sign or en-dash before digits to a math-mode minus sign. (We don't just use `$-`⟨*digits*⟩`$` because we don't have the main font set up for math mode, so that would make all the negative numbers show up in Computer Modern which would look weird. Having just the minus sign in that font is wrong too but is harder to really notice.

Unicode characters
: Unicode open and closing single and double quotes, hyphens, en-dashes, and em-dashes are converted to their equivalent TEX codes.

Local Transformations

There may often be campaign-specific local transformations needed such as italicizing certain Elvish words or converting DCs between editions of Pathfinder, etc. For these, create a file called `~/.gma/modcleanup.map` and write local substitutions, one per line, as needed. Each of these has the form

/⟨*regex*⟩/⟨*replacement*⟩/

The ⟨*regex*⟩ string is a regular expression matched against the text read from the input, and for every occurrence of a string matching that pattern, it will be replaced by the ⟨*replacement*⟩ string. The delimiter characters which appear before, after, and between these strings (shown here as "/") may be any single character; choose one that isn't needed for either string.

The substitutions will be carried out in the order listed in the file for each line of input before any built-in substitutions the script would normally make.

STANDARD PFCOMMON MACROS

For reference, this script assumes the use of my baseline set of LATEX macros, along with the rpgtex document style by Brian Criswell, et al. This section

documents the macros in my standard set, since this script assumes at least some of them are present and they don't have their own manpage yet anyway so this is where they're currently documented.

`\ar`
: Typeset AR.

`\area{`⟨*id*⟩`}`
: Typesets an area ID as " area ⟨*id*⟩" with the ⟨*id*⟩ in bold.

`\area*{`⟨*id*⟩`}`
: Typesets just the bolded ⟨*id*⟩ all by itself.

`\Area{`⟨*id*⟩`}`
: As above but capitalizes the word Area.

`\areas{`⟨*id1*⟩`}{...}{`⟨*id2*⟩`}`
: Typesets range of area ids as " areas ⟨*id1*⟩...⟨*id2"*⟩

`\Areas{`⟨*id1*⟩`}{...}{`⟨*id2*⟩`}`
: As above but capitalized.

`\areax{`⟨*name*⟩`}{`⟨*id*⟩`}`
: Allows for custom wording to be used instead of just "area".

`\attr{`⟨*name*⟩`}{`⟨*value*⟩`}`
: Used in some environments (see below) where various named attributes are listed. This shows the attribute ⟨*name*⟩ in bold, followed by the ⟨*value*⟩ with a non-breakable space between them. It also specifies that there should be no paragraph indent at the start of the attribute name.

`\attr*{`⟨*name*⟩`}{`⟨*value*⟩`}`
: As above, but the ⟨*name*⟩ and ⟨*value*⟩ are run together with no space between them (which allows you to explicitly indicate how you want to separate them).

`\attr!{`⟨*name*⟩`}{`⟨*value*⟩`}`
: As above, but ⟨*name*⟩ and ⟨*value*⟩ have a regular (breaking) space between them.

`\B{`⟨*text*⟩`}`
: Typesets ⟨*text*⟩ in boldface.

`\bonusfeat`
: Typeset a superscript B to indicate that a feat is a bonus feat.

`\Combat:` ⟨*text*⟩...
: Starts paragraph with bold **Combat:** lead-in text. *Does not start a paragraph break on its own.*

`\Creature:` ⟨*text*⟩...
: Starts paragraph with bold **Creature:** lead-in text. *Does not start a paragraph break on its own.*

`\Creatures:` ⟨*text*⟩...
: Starts paragraph with bold **Creatures:** lead-in text. *Does not start a paragraph break on its own.*

`\Development:` ⟨*text*⟩...
: Starts paragraph with bold **Development:** lead-in text. *Does not start a paragraph break on its own.*

`\DC{`⟨*level*⟩`}{`⟨*dc2*⟩`}`
: Convert a skill check difficulty class ⟨*dc2*⟩ designed for a Pathfinder 2nd Edition character of the given ⟨*level*⟩ to the equivalent DC for Pathfinder 1st Edition, and typeset with with "DC" in front of that number. As a special case, if ⟨*level*⟩ is "*", make no conversion, just use ⟨*dc2*⟩ as given.

`\DC*{`⟨*level*⟩`}{`⟨*dc2*⟩`}`
: As above but only print the converted DC number itself.

`\dm`
: Typeset DM.

`\domainspell`
: Typeset a superscript D to indicate that a spell is a domain spell.

`\gm`
: Typeset GM.

`\Haunt:` ⟨*text*⟩...
: Starts paragraph with bold **Haunt:** lead-in text. *Does not start a paragraph break on its own.*

`\Hazard:` ⟨*text*⟩...
: Starts paragraph with bold **Hazard:** lead-in text. *Does not start a paragraph break on its own.*

`\Hazards:` ⟨*text*⟩...
: Starts paragraph with bold **Hazards:** lead-in text. *Does not start a paragraph break on its own.*

`\ii{`⟨*number*⟩`}`
: Attach item ID number to an item. Typesets ⟨*number*⟩ as a superscripted sans-serif number.

`\LevelBox{`⟨*number*⟩`}`
: Typset ⟨*number*⟩ in a large colored box. This is usually used in the `AdvancementTable` environment.

`\magicitem{`⟨*name*⟩`}`
: Typeset a magic item name.

`\mc{`⟨*text*⟩`}`
: Set text in medium-caps size font (does not capitalize the text).

`\ModuleID{`⟨*id*⟩`}`
: Typeset the module id number in a colored box (typically used on the front cover).

`\MonsterList{`⟨*name*⟩`}{`⟨*qty*⟩`}{`⟨*hp*⟩`}{`⟨*book*⟩`}{`⟨*page*⟩`}`
: Typesets a list of monsters such as would appear as part of a room description. This doesn't go into detail, just says how many of a given type are here, and references where the details of the creatures may be found, in published books or other pages of the module. The ⟨*qty*⟩ field may be left blank, as may ⟨*book*⟩ or ⟨*page*⟩. ⟨*Book*⟩ is always typeset in Italics.

`\NL`
: Used in environments where there are attributes which sometimes break lines strangely, this inserts a new line break and asks that the next line not be indented.

`\mythicspell`
: Typeset a superscript M to indicate that a spell is a mythic spell.

`\Note:` ⟨*text*⟩...
: Starts paragraph with bold **Note:** lead-in text. *Does not start a paragraph break on its own.*

`\NPC{`⟨*name*⟩`}{`⟨*notes*⟩`}`
: Indicates information about an NPC. Bolds their ⟨*name*⟩ and follows it with the ⟨*notes*⟩ about them in parentheses (if any; that parameter may be left empty).

`\Object{`⟨*name*⟩`}{`⟨*thickness*⟩`}{`⟨*hardness*⟩`}{`⟨*hp*⟩`}{`⟨*break DC*⟩`}`
: Typesets a description of an object found here with breakage stats. The ⟨*thickness*⟩ and ⟨*break DC*⟩ fields may be left blank.

`\Object*{`⟨*name*⟩`}{`⟨*thickness*⟩`}{`⟨*extra*⟩`}{`⟨*hardness*⟩`}{`⟨*hp*⟩`}{`⟨*break DC*⟩`}`

: As above, but adds extra text which may appear after the thickness. In this case, ⟨*extra*⟩ and ⟨*break DC*⟩ may be left blank.

`\ooc{`⟨*text*⟩`}`
: Typeset the ⟨*text*⟩ as out-of-character commentary (something not necessarily in-universe for the module's story). This renders it in steel blue text.

`\OGL{`⟨*original published document*⟩`}{`⟨*copyright year*⟩`}{`⟨*author(s)*⟩`}`
Print the Open Gaming License text.

`\ParB{`⟨*heading*⟩`}`
Begins a new paragraph, starting with the text "heading" in boldface.

`\ParI{`⟨*heading*⟩`}`
Begins a new paragraph, starting with the text "heading" in Italics.

`\PFItemHeader{`⟨*name*⟩`}{`⟨*x*⟩`}`
Deprecated. Use `PFBlock` instead.

`\PFsec{`⟨*title*⟩`}`
Start a section within a Pathfinder block (generally used in a `PFBlock` environment or another environment based on it). This sets the ⟨*title*⟩ between 1-point horizontal rules to introduce a new section of the block.

`\PFtrap{`⟨*name*⟩`}{`⟨*cr*⟩`}{`⟨*xp*⟩`}{`⟨*type*⟩`}{`⟨*perception DC*⟩`}{`⟨*disable DC*⟩`}{`⟨*trigger*⟩`}{`⟨*reset*⟩`}{`⟨*effect*⟩`}`

Typesets the description of a trap using a standard Pathfinder 1E format block. The ⟨*effect*⟩ may be a large block of text describing the full effects of the trap's operation.

`\sourcemarker{`⟨*source*⟩`}`
Typesets the ⟨*source*⟩ name as a superscript to indicate the source book something came from (e.g. `\sourcemarker{APG}` for the Advanced Player's Guide).

`\speech{`⟨*text*⟩`}`
Typeset ⟨*text*⟩ as a direct quote of a character's spoken words. To make it easier to find dialog in the module, this is set in bold blue type.

`\speech*{`⟨*text*⟩`}`
Variation which sets the text in teal (to easily distinguish different speakers).

`\speech!{`⟨*text*⟩`}`
Variation which sets the text in red (to easily distinguish different speakers).

`\speech@{`⟨*text*⟩`}`
Variation which sets the text in plain bold type.

`\spell{`⟨*name*⟩`}`
Typeset a spell name.

`\strong{`⟨*text*⟩`}`
Typeset ⟨*text*⟩ in boldface.

`\tactical{`⟨*a*⟩`}{`⟨*b*⟩`}{`⟨*EL*⟩`}`
: Start a new chapter for a tactical encounter section titled "⟨*a*⟩: ⟨*b*⟩" with internal reference label `tactical-`⟨*b*⟩ and with encounter level ⟨*EL*⟩.

`\Tactics:` ⟨*text*⟩...
: Starts paragraph with bold **Tactics:** lead-in text. *Does not start a paragraph break on its own.*

`\Trap{`⟨*name*⟩`}{`⟨*cr*⟩`}{`⟨*x*⟩`}{`⟨*x*⟩`}{`⟨*x*⟩`}{`⟨*reset*⟩`}{`⟨*x*⟩`}{`⟨*perception*⟩`}{`⟨*disable*⟩`}`

: Deprecated. Use `\PFtrap` instead.

`\TrapNote:` ⟨*text*⟩...
: Starts paragraph with bold **Trap:** lead-in text. *Does not start a paragraph break on its own.*

`\Treasure:` ⟨*text*⟩...
: Starts paragraph with bold **Treasure:** lead-in text. *Does not start a paragraph break on its own.*

`\work{`⟨*title*⟩`}`
: Typeset the name of a published work.

`\XP Award:` ⟨*text*⟩...
: Starts paragraph with bold **XP Award:** lead-in text. *Does not start a paragraph break on its own.*

`\XXX{`⟨*text*⟩`}`
: Highlight ⟨*text*⟩ as something that needs to be resolved later.

Environments

`AdvancementTable`
: This environment is a specialized kind of table that is designed to show what the party's expected level progression through the module should be. An example usage might be:

```
\begin{AdvancementTable}
 \LevelBox{1} & The PCs begin at level 1.\\\cline{2-2}
 \LevelBox{2} & The PCs should reach level 2 before
                confronting the goblins.\\\cline{2-2}
 \LevelBox{3} & The PCs should reach level 3 by the end
                of the adventure.
\end{AdvancementTable}
```

`AdvancementTableII`
: As above but allows room for two-digit level numbers.

`PFArtifact*{`⟨*name*⟩`}{`⟨*aura*⟩`}{`⟨*CL*⟩`}{`⟨*slot*⟩`}{`⟨*minor / major*⟩`}{`⟨*weight*⟩`}`

: Just like `PFMagicItem*` but for artifacts. Instead of using `\construction*`, however, use `\destruction*` (no arguments), which introduces a section detailing how the artifact may be destroyed.

`PFBlock{`⟨*title*⟩`}`
: Everything enclosed between `\begin{PFBLock}{`⟨*title*⟩`}` and `\end{PFBlock}` is a block set in a generic Pathfinder 1E style that is used for various things like creatures, items, settlements, weapons, pretty much anything being described that is out of the main flow of the book's text. The text is set in sans-serif font to distinguish it visually from the main running text of the rest of the module, and is started with a colorful bar with upper-cased ⟨*title*⟩ across the top. You can use this environment to make custom blocks, but other environments documented here are based on `PFBlock` which do more specific kinds of things.

`PFHaunt{`⟨*name*⟩`}{`⟨*cr*⟩`}{`⟨*xp*⟩`}{`⟨*description*⟩`}{`⟨*CL*⟩`}{`⟨*notice*⟩`}{`⟨*hp*⟩`}{`⟨*trigger*⟩`}{`⟨*reset*⟩`}`

: Begin a `PFBlock` describing a haunt. The interior of the environment works like a `description` environment, but use `\haunt{`⟨*name*⟩`}` to introduce items. Typical items include `\haunt{Effect}` and `\haunt{Destruction}`.

`PFMagicItem{`⟨*name*⟩`}{`⟨*arua*⟩`}{`⟨*CL*⟩`}{`⟨*slot*⟩`}{`⟨*price*⟩`}{`⟨*weight*⟩`}`
: While I prefer `PFMagicItem*` instead, this is merely a simpler form that uses `\attr` style headers instead of the tabular format.

`PFMagicItem*{`⟨*name*⟩`}{`⟨*aura*⟩`}{`⟨*CL*⟩`}{`⟨*slot*⟩`}{`⟨*price*⟩`}{`⟨*weight*⟩`}`
: Begin a `PFBlock` describing a magic item. This sets up a Pathfinder-style header table with the provided stats in it, then allows for more or less free-form text within the rest of the environment. Use `\attr` macros where appropriate. At the bottom, use `\construction{`⟨*requirements*⟩`}{`⟨*cost*⟩`}` to introduce the construction section that will then describe how the item is constructed, or the alternate form `\construction*{`⟨*requirements*⟩`}{`⟨*cost*⟩`}` which formats those in a little table instead of the more austere style.

`PFMonster{`⟨*name*⟩`}{`⟨*cr*⟩`}{`⟨*xp*⟩`}{`⟨*alignment*⟩ ⟨*size*⟩ ⟨*type*⟩`}`
: Typesets a monster entry in a `PFBlock`. Note that the fourth parameter is a free-form text field but typically contains the creature's alignment, size, and type like "N Large construct". Inside this environment, use `\attr` and `\NL` macros as needed (and you will), along with the following macros and environments specially designed for monsters, typically in this order. Note that it may be necessary to

end some lines with % to eat newlines in the source file to avoid extra spaces getting into the output until I have a chance to tweak the macros better.

`\senses{`⟨*init*⟩`}{`⟨*senses*⟩`}`
: List the creature's initiative modifier and all their senses such as Perception, Darkvision, and so forth. Follow this with any additional `\attr` entries that go before the defense section.

`\defense{`⟨*ac*⟩`}{`⟨*hp*⟩`}{`⟨*fort*⟩`}{`⟨*refl*⟩`}{`⟨*will*⟩`}{`⟨*additional*⟩`}`
: Start the defense section. Specify armor class, hit points, saving throws, plus any additional special defenses the creature may have (this last field may be left blank). Follow this with any additional `\attr` entries that go before the offense section.

`\offense{`⟨*speed*⟩`}{`⟨*melee*⟩`}{`⟨*ranged*⟩`}{`⟨*space*⟩`}{`⟨*reach*⟩`}{`⟨*special*⟩`}`
: Start the offense section. Specify movement speed(s), melee and ranged attack forms, along with the creature's space and reach distances. If there are special attacks, list them as well. The ⟨*ranged*⟩ and ⟨*special*⟩ fields may be left blank. Follow this with any additional `\attr` entries that go before the statistics section.

`\statistics{`⟨*str*⟩`}{`⟨*dex*⟩`}{`⟨*con*⟩`}{`⟨*int*⟩`}{`⟨*wis*⟩`}{`⟨*cha*⟩`}{`⟨*bab*⟩`}{`⟨*cmb*⟩`}{`⟨*cmd*⟩`}`
: Start the statistics block, providing the creature's abilitiy scores, base attack bonus, combat maneuver bonus and combat maneuver defense. Follow this with any additional `\attr` entries that go before the spells, special abilities, or feats sections.

`\begin{SpecialAbilities}...\end{SpecialAbilities}`
: This environment is a special list environment similar to `description` which is used to describe the creature's special abilities. Each ability is described by introducing it with `\ability{`⟨*name*⟩`}` which starts a new special ability entry. The text which follows (which may be multiple paragraphs if necessary) describes that ability.

  Normally, this is set in a section which is actually titled "SPECIAL ABILIITIES", but this environment is useful for other things too, so there are two variant forms you can use. The `SpecialAbilities*` environment works the same but just doesn't include the section title at all (allowing you to place one of your own with `\PFsec`—or even omitting one entirely), while the `SpecialAbilities!` environment not only omits the section title but also removes the normal spacing between the ability name and the start of the ability text for each of the `\ability` entries. This makes them useful for typesetting spells and spell-like abilities for monster entries.

  For example, a monster may have this at the end of its offense section:

```
\begin{SpecialAbilities*}
 \ability{Spell-like Abilities} CL 12th,
          concentration +21 (+25 casting defensively)
 \begin{SpecialAbilities!}
  \ability{At will---}\spell{chill touch},
                      \spell{command undead}
  \ability{3/day---}\spell{animate dead}
  \ability{1/day---}\spell{slay living}
 \end{SpecialAbilities!}
 \ability{Cleric Spells Prepared} CL 10th,
          concentration +14 (+18 casting defensively)
 \begin{SpecialAbilities!}
  \ability{5th---}\spell{mass cure light wounds}, ...
...
 \end{SpecialAbilities!}
\end{SpecialAbilities*}
```

Whereas the usual special ability section would look like this:

```
\begin{SpecialAbilities}
 \ability{Poison (Ex)} Injury---bite;
  \emph{save} Fort DC~22;
  \emph{duration} 1/round for 4 rouunds;
  \emph{effect} 1d4 Wisdom damage;
  \emph{cure} 1 save. The save DC is Constitution
              based.
\end{SpecialAbilities}
```

`\feats{`⟨*list*⟩`}`
: List all the feats the creature has.

`\skills{`⟨*list*⟩`}`
: List all the skills and associated bonuses the creature has.

`\languages{`⟨*list*⟩`}`
: List all the languages spoken by the creature.

`\ecology{`⟨*environment*⟩`}{`⟨*organization*⟩`}{`⟨*treasure*⟩`}`
: List the environment, organization, and typical treasure for this creature type.

`PFSettlement{`⟨*name*⟩`}{`⟨*alignment*⟩ ⟨*size*⟩ ⟨*type*⟩`}`
: Typesets a settlement block. Note that the second parameter is a free-form text field but typically consists of the town's average aligment, size, and type, e.g., "NG small town". Inside this environment, use

`\attr` macros as needed, along with these macros specially designed for settlements, typically in this order:

`\qualities{`⟨*corruption*⟩`}{`⟨*crime*⟩`}{`⟨*economy*⟩`}{`⟨*law*⟩`}{`⟨*lore*⟩`}{`⟨*society*⟩`}{`⟨*qualities*⟩`}{`⟨*dang*

Provides the modifier values for the named stats and the named quality list for the settlement.

`\demographics{`⟨*government*⟩`}{`⟨*population*⟩`}`
List the type of government and population (overall and breakdown if desired).

`\notablenpcs{`⟨*text*⟩`}`
List the notable NPCs in town, using the `\NPC` macro, usually with `\NL` between each.

`\marketplace{`⟨*base*⟩`}{`⟨*limit*⟩`}{`⟨*spellcasting*⟩`}{`⟨*minor*⟩`}{`⟨*medium*⟩`}{`⟨*major*⟩`}`

List the market limits and ranges of what's available for hire and purchase.

`PFSpell{`⟨*name*⟩`}{`⟨*school*⟩`}{`⟨*level*⟩`}{`⟨*casting time*⟩`}{`⟨*components*⟩`}`
Typesets a spell description in a `PFBlock`. Use any appropriate typesetting commands inside the environment to describe the spell as desired. The parameters to the environment itself set up the standard header lines. Additional ones may be added using `\attr` macros.

Counters

The counter `CampaignStartYear` is set to the year the campaign begins. Other counters are defined for significant events in the story and calculated relative to that main counter, so the adventure can be moved to any starting date and will bring the dependent dates in its timeline along with it.

AUTHOR

Steve Willoughby / steve@madscience.zone.

gma-namegen(6)

NAME

gma namegen – generate random Golarion-style names

SYNOPSIS

```
gma namegen -h
gma namegen -l
gma namegen -d -c ⟨culture⟩
gma namegen [-s] [-c ⟨culture⟩] [-g ⟨gender⟩] [-M ⟨lmax⟩] [-m ⟨lmin⟩]
[-n ⟨num⟩] [-S ⟨text⟩]
gma namegen --help
gma namegen --list-cultures
gma namegen --describe --culture ⟨culture⟩
gma namegen --culture ⟨culture⟩ [--surnames] [--gender ⟨gender⟩]
[--start ⟨text⟩] [--num ⟨n⟩] [--lmin ⟨min⟩] [--lmax ⟨max⟩]
```

DESCRIPTION

This program generates random names which thematically fit in with various cultures of Golarian.

OPTIONS

The following options control the behavior of `namegen`.

`-c`, `--culture` ⟨*culture*⟩
: Specify the culture on which to base the names.

`-d`, `--describe`
: Describe the attributes of the specified culture and exit.

`-g`, `--gender` ⟨*gender*⟩
: Generate names for the specified gender. The list of genders available depends on the culture, but the usual set of values which may be given includes:

  `F` Female given names

  `M` Male given names

  `S` Surnames

  The list of genders available may be found via the `--describe` option. The default is `F`.

`-h`, `--help`
: Print the basic usage information for the program and exit.

`-l`, `--list-cultures`
: Print the list of all available cultures and exit.

`-M`, `--lmax` ⟨*max*⟩
: Specify the maximum length for names.

`-m`, `--lmin` ⟨*min*⟩
: Specify the minimum length for names.

`-n`, `--num` ⟨*n*⟩
: Print ⟨*n*⟩ names. The default is 10.

`-S`, `--start` ⟨*text*⟩
: Names must begin with the specified ⟨*text*⟩.

`-s`, `--surnames`
: Generate surnames for each given name (if supported by the culture).

SEE ALSO

`gma-pfnamegenerator`(3).

AUTHOR

Steve Willoughby / steve@madscience.zone. This is a port of an earlier implementation by John Mechalas.

HISTORY

This utility was first created 05-Mar-2020.

gma-papertokens(6)

NAME

gma papertokens – generate LaTeX document with scaled creature token images

SYNOPSIS

`gma papertokens` [`-o` ⟨*output*⟩`.tex`] [`-`⟨*size*⟩] [`-`⟨*qty*⟩] ⟨*inputfile*⟩ ...

DESCRIPTION

You have your adventure all prepped and ready to play. Map and creature token files are prepared and loaded into the server for use with the `gma-mapper`(6) program. Your encounters are written up in a GMO file to automate running combat. But then, disaster strikes! (Or perhaps you planned it this way from the beginning.) You need to play the game the old-fashioned, dare we say, "unplugged" way, on a physical table-top with miniatures placed on a large paper map. But you don't have miniature figures for all your monsters. You do, however, have your token image files you were going to use with `mapper`. If only you could easily print those out as paper tokens you can place on your paper map on the table.

If this is the situation you find yourself in, you are reading the right document.

`Papertokens` reads a set of map creature token files such as those already processed by `gma-rendersizes`(6), and emits a LaTeX document to its standard output. If this document is then typeset by a program such as `pdflatex`(1), it will produce a PDF document with image tokens scaled for use with a standard 1 inch to 5 foot grid size battle mat for table-top play without a computer.

OPTIONS

The following options, each beginning with a hyphen (`-`), may appear before any ⟨*inputfile*⟩ names. Those which specify size and quantities may also appear between ⟨*inputfile*⟩ names to affect the input files which appear after them in the list.

`-o` ⟨*output*⟩`.tex`
: Write the LaTeX output to the named ⟨*output*⟩ file instead of the standard output. This is recommended to avoid any diagnostic messages that GMA might produce on standard output from getting mixed up in the LaTeX document source. To avoid accidental overwriting of image files or confusion as to the nature of the output file, its name must end with "`.tex`".

-⟨*size*⟩
: All ⟨*inputfile*⟩ names which follow are for creatures of the given ⟨*size*⟩ (default is medium-size). The ⟨*size*⟩ value is the lower-case size category code, so the actual option names as written on the command line would be `-f`, `-d`, `-t`, `-s`, `-m`, `-l`, `-h`, `-g`, or `-c`, respectively, for size categories fine, diminutive, tiny, small, medium, large, huge, gargantuan, or colossal.

-⟨*qty*⟩
: (where ⟨*qty*⟩ is an integer) All ⟨*inputfile*⟩ names which follow should be printed ⟨*qty*⟩ times.

FILENAME PATTERNS

The file name of the image is assumed to conform to GMA's creature token image naming conventions. Names which begin with an alphabetic character or underscore are processed as-is (on the assumption that they may not have been created by `gma-rendersizes`(6) but were still correctly prepared for this purpose). Otherwise, names beginning with the prefix characters `#`, `=`, `-`, and `+` are interpreted correctly (as representing skins 0, 1, 2, and 3, respectively, of the creature), as are those with the general skin-specifier prefix `^`⟨*n*⟩`^` for any arbitrary skin number ⟨*n*⟩.

Filenames which conform to the above rules but have a percent sign (`%`) prefix at the start of the name are assumed to be the version of the creature token to use when the creature is dead. This will be drawn with a large red "X" through the token. (When printing these out, you may wish to glue these on the back of the regular version of the tokens, so you can flip the token over to indicate that it was killed on the battlefield.)

Filenames starting with other characters, including the standard GMA prefix `!`, are ignored.

The rest of the pathname, including directory names, should be alphanumeric to avoid confusing LaTeX with special characters in their names.

Printing from Prepared GMO Files

If an ⟨*inputfile*⟩ name ends with "`.gmo`", it is read as a GMA monster encounter definition file, and all of the creature tokens it describes, at the sizes and quantities listed in the file, are added to the paper token document. In this case, the image files must be in the current directory, or in a subdirectory whose name is a single letter matching the size category of the creature (in either upper- or lower-case).

For example, if an `image_tag` field identified a medium-sized monster token called `Orc`, the corresponding image file would be expected to be named `^0^Orc.png`, `#Orc.png`, or `Orc.png`, located in the current directory or a subdirectory named `m` or `M`; if the orc has a disguise listed in the file as another medium-size skin, that image would need to be in a file called `^1^Orc.png`

or `=Orc.png` in the same directories (or with image file suffix `.gif`, `.jpeg`, or `.jpg`).

In addition, if there exist any files called `%Orc.png`, `%^0^Orc.png`, `%#Orc.png`, `%=Orc.png`, `%^1^Orc.png`, etc., those would be printed as the "dead" versions of the Orc's token.

NOTA BENE

Keep in mind that `papertokens` does not *generate* any image files (not even to make "dead" variations of creatures). It only creates a scaled sheet full of creature tokens from already-existing image files.

EXAMPLES

To print two medium Orc images and one huge Golem from files in the current directory:

```
$ gma papertokens -o monsters.tex -2 Orc.png -h Golem.png
$ pdflatex monsters
```

To print all the monsters whose files are in the current directory and size-code-named subdirectories from the GMO file called `monsters.gmo`:

```
$ gma papertokens -o monsters.tex monsters.gmo
$ pdflatex monsters
```

In both examples, the file `monsters.pdf` is produced which you can then print onto paper.

SEE ALSO

`gma-character-record`(5), `gma-monster-sheet`(5), `pdflatex`(1), `gma-rendersizes`(6).

AUTHOR

Steve Willoughby

BUGS

This is a pretty simple-minded convenience script, which is probably easy to confuse with weird filenames.

Why LaTeX? Because this was a quick script to do a simple job I needed to do, and I always have LaTeX installed on my systems, and it conveniently provides for putting a bunch of image files at explicit scaled sizes together on a document, providing the quickest path from point A to point B at the time.

If you process GMO files as inputs, `papertokens` invokes the GMA code to interpret that file, which may cause stray messages to appear on the standard output which LaTeX won't know what to do with. Hence the `-o` option.

`Papertokens` does not support custom creature sizes such as "`C=100`" that do not match a standard size category.

gma-pfcommon(7)

See gma-modcleanup(6), p. 504.

gma-preferences(6)

NAME

gma preferences – Edit and query GMA tool preferences settings

SYNOPSIS

```
gma preferences [-h] [-g ⟨var⟩] [-n ⟨network⟩] [-p ⟨path⟩] [-w
⟨world⟩]
gma preferences --help

gma preferences [--get ⟨var⟩] [--network ⟨network⟩]
[--preferences ⟨path⟩] [--world ⟨world⟩]
```

DESCRIPTION

All of the GMA tools use a common preferences setting file, by default stored in `~/.gma/preferences.json`, to guide how they should operate for the user's campaign. (The `gma-mapper`(6) program uses its own separate preferences file. For that matter, while theoretically all other GMA tools *could* use the central preferences file, not all of them actually do.)

While individual GMA programs, including `gma`(6) itself, have the ability to let you access the preferences settings from inside those programs, the `gma preferences` program documented here is a stand-alone utility that exists to edit those preferences without starting another tool first. (This is necessary before starting `gma` for the very first time, to set up the initial set of operating parameters for it.)

Without other options, a GUI preferences editor window is displayed, with which you can review and edit the settings. See below for a description of each such setting.

OPTIONS

The following command-line options may be used to alter the default behavior of `gma preferences`:

`-g, --get` ⟨*var*⟩
: Retrieve the value of the named variable ⟨*var*⟩, printing it to the standard output before exiting. Note that some values are arranged hierarchically. For the purposes of the `--get` option, this is notated using dots between each level. For example, `appearance.dark_mode` or `appearance.scaling` are both appearance-related values.

  If you run `gma preferences --get appearance.dark_mode`, it will print something like "`True`", to indicate that dark mode is enabled. However, if you only ask for the higher-level name, all values belonging to

that level will be printed. So invoking `gma preferences --get appearance` would print something like "`{'dark_mode': True, 'scaling': 1.0}`".

`-h, --help`
Prints a summary of command-line options and exits.

`-n, --network` network
Select the named ⟨*network*⟩ as the current network for the purposes of the operation of this program. This does not persistently change the current network selection unless the preferences are saved to disk during this session.

`-w, --world` world
Select the named ⟨*world*⟩ as the current world for the purposes of the operation of this program. This does not persistently change the current world selection unless the preferences are saved to disk during this session.

SETTINGS

The settings are divided by topic into a number of tabs as shown in the GUI editor. We will describe them in the same way. For each, the name as shown in the GUI interface is listed first, followed by the internal name as specified to the `--get` option.

General

These are settings that don't fit into a more specific category and/or apply globally to GMA at a fundamental level.

Core SRD Database Path `(core_db)`
The pathname to the GMA SRD (aka core) database used to get game world data. This is only used by GMA versions 7.0 and later. Earlier GMA versions will ignore this value.

Appearance

These settings control the visual appearance of the GUI.

Dark color theme `(appearance.dark_mode)`
If checked, dark-mode colors are selected for GUI elements.

Visual scaling factor `(appearance.scaling)`
Normally this is set to `1.0` but this may be changed to adjust for high-resolution monitors if the GUI elements are drawn too small.

Worlds

This tab defines all of the campaign-world-dependent settings. At the top of the window is a list of all defined worlds. New worlds may be added to the list by clicking the "Add New World..." button (which starts with all default values for the world), or the "Copy" button (which starts with a copy of the currently-selected world).

Whichever world name on the list is selected (by clicking the mouse on it) is the default world the GMA tools will assume, and all of the rest of the values shown on this window will be those which apply to that game world.

The currently selected world, and all settings associated with it, may be deleted by clicking on the "Delete" button.

The currently-selected game world name may be accessed via the `--get` option using the name `current_world`. The set of all worlds in their entirety may be retrieved by the name `worlds`.

Individual world settings may be accessed as `worlds.current.`⟨*var*⟩ to get the value of ⟨*var*⟩ for the currently-selected world, or `worlds.`⟨*name*⟩`.`⟨*var*⟩ for the world profile with the given ⟨*name*⟩. (Thus, in the following lists, ⟨*x*⟩ may be the name of a world or the word "`current`".)

Calendar Type (`worlds.`⟨*x*⟩`.calendar_type`)
: The name of the calendar system in use for the campaign.

Module Encounter (.gmo) Path (`worlds.`⟨*x*⟩`.module`)
: The pathname of the encounter `.gmo` file to use for pre-generated encounters.

Blur HP (%) (`worlds.`⟨*x*⟩`.blur_hp`)
: The percentage by which to round off creature hit points as displayed to the players, so they don't see immersion-breaking levels of detail. If 0, the true hit points are reported.

Campaign Database Path (`worlds.`⟨*x*⟩`.db_name`)
: The path to the sqlite3 database where campaign information is tracked by GMA. Note that this is the *full actual* pathname of the database file. The older configuration file used by GMA only specified the initial part of the filename, with the rest automatically supplied by GMA at runtime.

Campaign Display Name (`worlds.`⟨*x*⟩`.display_name`)
: The name of your campaign as displayed by the various tools.

GM Password (`worlds.`⟨*x*⟩`.password`)
: The password used to log in to the GMA server as the GM.

Initiative Backup Path (`worlds.`⟨*x*⟩`.initiative_backup_path`)
: At various intervals during combat, GMA will save the initiative list to this file, to guard against data loss if GMA should crash mid-combat.

Static Initiative List (`worlds.`⟨*x*⟩`.initiative_seed`)
: This is the list of creatures which should always be added to the initiative list when GMA starts. (In other words, the list is pre-seeded with these creatures before introducing the NPCs and monsters they will encounter.) Normally, this will be the player character party.

  In the GUI, this is a list that functions just like the list of worlds as described above. Each entry on the list is the name of a character as they will appear on the `mapper` tool. When a character is selected in the list box, their stats in the following fields are available to view and edit.

  When retrieving these values with the `--get` option, you may specify them as `worlds.`⟨*x*⟩`.initiative_seed.`⟨*name*⟩`.`⟨*var*⟩ for a value of a character with the given ⟨*name*⟩, or as `worlds.`⟨*x*⟩`.initiative_seed.`⟨*i*⟩`.`⟨*var*⟩ for a value from character #⟨*i*⟩ in the list (where ⟨*i*⟩ starts with 0 for the first one in the list). Thus, in the list below, ⟨*y*⟩ may be a character name or position number in the list.

  Their name is available as `worlds.`⟨*x*⟩`.initiative_seed.`⟨*name*⟩`.name` or `worlds.`⟨*x*⟩`.initiative_seed.`⟨*i*⟩`.name`.

  Dexterity (`worlds.`⟨*x*⟩`.initiative_seed.`⟨*y*⟩`.dex`)
  : The character's Dexterity score.

  Non-Dex Initiative (`worlds.`⟨*x*⟩`.initiative_seed.`⟨*y*⟩`.init_adj`)
  : The character's initiative modifier, not counting the modifier from Dexterity.

  Constitution (`worlds.`⟨*x*⟩`.initiative_seed.`⟨*y*⟩`.con`)
  : The character's Constitution score.

  Full Hit Points (`worlds.`⟨*x*⟩`.initiative_seed.`⟨*y*⟩`.hp`)
  : The character's maximum hit point total (when at full health).

  Blur HP (%) (`worlds.`⟨*x*⟩`.initiative_seed.`⟨*y*⟩`.blur_hp`)
  : This overrides the world-global blur HP value for this character only.

  Player Character? (`worlds.`⟨*x*⟩`.initiative_seed.`⟨*y*⟩`.is_pc`)
  : If checked, this character is a PC; otherwise it's an NPC or monster.

  Initiative Dice (`worlds.`⟨*x*⟩`.initiative_seed.`⟨*y*⟩`.die_spec`)
  : The dice to use when rolling initiative for this character, if not the standard "d20". For example, a character may wish to set this to "`d20 best of 2`". Note that applicable modifiers will be added to this string.

Static Caster List (`worlds.`⟨*x*⟩`.caster_levels`)
: This is the list of creatures who have caster levels. They are usually a subset of the names in the initiative list, but they need not be. These

are just the characters you want to have caster level calculations readily available from spell description windows and the like.

In the GUI, this is a list that functions just like the list of worlds as described above. Each entry on the list is the name of a character. When a character is selected in the list box, their caster stats in the following fields are available to view and edit.

When retrieving these values with the `--get` option, you may specify them as `worlds.`⟨*x*⟩`.caster_levels.`⟨*name*⟩`.`⟨*var*⟩ for a value of a character with the given ⟨*name*⟩, or as `worlds.`⟨*x*⟩`.caster_levels.`⟨*i*⟩`.`⟨*var*⟩ for a value from character #⟨*i*⟩ in the list (where ⟨*i*⟩ starts with 0 for the first one in the list). Thus, in the list below, ⟨*y*⟩ may be a character name or position number in the list.

Their name is available as `worlds.`⟨*x*⟩`.caster_levels.`⟨*name*⟩`.name` or `worlds.`⟨*x*⟩`.caster_levels.`⟨*i*⟩`.name`.

Caster Level (`worlds.`⟨*x*⟩`.caster_levels.`⟨*y*⟩`.cl`)
: The character's total caster level.

Networks

This tab defines all of the network profiles you may use to connect to a GMA server. At the top of the window is a list of all defined networks. New networks may be added to the list by clicking the "Add New network..." button (which starts with all default values for the network), or the "Copy" button (which starts with a copy of the currently-selected network).

Whichever network name on the list is selected (by clicking the mouse on it) is the default network the GMA tools will use, and all of the rest of the values shown on this window will be those which apply to that network profile.

The currently selected network, and all settings associated with it, may be deleted by clicking on the "Delete" button.

The currently-selected network profile name may be accessed via the `--get` option using the name `network_profile`. The set of all network profiles in their entirety may be retrieved by the name `networks`.

Individual network settings may be accessed as `network.current.`⟨*var*⟩ to get the value of ⟨*var*⟩, or `network.`⟨*name*⟩`.`⟨*var*⟩ for the network profile with the given ⟨*name*⟩.

Server Hostname (`network.`⟨*x*⟩`.hostname`)
: The name or IP address of the server.

Server TCP Port (`network.`⟨*x*⟩`.port`)
: As the name implies.

(unlisted)

The following value is not available via the GUI tool but may be queried via the `--get` option.

`gma_preferences_file_version`
: The version number of the preferences file format. Any time the layout of the file is changed, usually by adding new setting values, this number is increased.

AUTHOR

Steve Willoughby / steve@madscience.zone.

gma-printchar(6)

NAME

gma printchar – generate player character sheets from datafiles

SYNOPSIS

```
gma printchar [-AhHZz] [-c ⟨color⟩] ⟨playerfile⟩ ...
gma printchar [--all-skills] [--color-scheme ⟨clr⟩]
[--compact-text] [--help] [--highlight-important-areas]
[--suppress-text] ⟨playerfile⟩ ...
```

DESCRIPTION

Takes the character information in the ⟨*playerfile*⟩, (which by convention has the filename suffix "`.gcr`") along with information stored in the GMA database for this character, and generates a PostScript character sheet for use during the game.

The PostScript reports are written to and output file with the same name as the input (⟨*playerfile*⟩), but with the suffix replaced with `.ps`.

OPTIONS

The following options may be used to control the behavior of the program. They may be given in the classic "short form" consisting of a single letter prefixed by a hyphen (e.g., "`-A`"), or in the newer "long form" will spelled-out option names prefixed by a double hyphen (e.g., "`--all-skills`"). Short-form options may be combined into a single argument (e.g., "`-AZ`" is equivalent to specifying both "`-A`" and "`-Z`").

Any parameters to an option may be specified in the argument immediately following the option (e.g., "`-c red`" or "`--color-scheme red`"), or combined with the option into a single argument (e.g., "`-cred`" or "`--color-scheme=red`").

`-A`, `--all-skills`
: Print all possible skills on the character sheet, rather than only those the character could possibly actually use, given their current training.

`-c`, `--color-scheme` ⟨*clr*⟩
: Use the color scheme ⟨*clr*⟩ for the output document. Currently defined colors include: `black`, `blue`, `green`, `red`, `orange`, and `violet`.

`-h`, `--help`
: Print usage information and exit.

`-H`, `--highlight-important-areas`
: To make it easier to work with the character record during the game

and draw your attention to the most significant data at a glance, highlight in yellow the most vital stats for the character such as ability modifiers, armor class, and hit points. This should also be easier for newer players who don't yet have a feeling for what things on the character sheet they need to start off understanding and which can be left to be explained as they come up.

`-z`, `--compact-text`
: Compress large text blocks, such as spell descriptions, into space-saving dense blocks of text. Some typographical features (e.g., tables) are omitted entirely.

`-Z`, `--suppress-text`
: Go beyond `-z` and actually suppress large text blocks entirely.

SEE ALSO

`gma`(6), `gma-character-record`(5).

AUTHOR

Steve Willoughby / steve@madscience.zone

HISTORY

This document describes version 4.1 of `printchar`.

gma-printmon(6)

NAME

gma printmon – generate monster encounter sheets from datafiles

SYNOPSIS

```
gma printmon [-adhZz] [-c ⟨color⟩] ⟨monsterfile⟩ ...
gma printmon [--help] [--append] [--color-scheme ⟨color⟩]
[--duplex] [--compact-text] [--suppress-text] ⟨monsterfile⟩ ...
```

DESCRIPTION

To prepare for an adventure, the GM sets up the monster encounters by writing a ".gmo" file as documented in gma-monster-sheet(5). This describes each encounter area, with details of the creatures encountered in that area.

To facilitate offline play (or to supplement the GMA tools with additional paper reference material), the printmon program generates PostScript run sheets in a manner similar to the way gma-printchar(6) prints character record sheets.

These include the most essential creature stat blocks and special abilities, and a row of checkboxes to tick off hit points as each creature takes damage. Hit points are randomly rolled for each creature, and names are randomly generated as detailed in the input file(s).

The PostScript reports are written to output files, each with the same name as the corresponding input file, but with the suffix replaced with .ps.

OPTIONS

The following options may be used to control the behavior of the program. They may be given in the classic "short form" consisting of a single letter prefixed by a hyphen (e.g., "-a"), or in the newer "long form" will spelled-out option names prefixed by a double hyphen (e.g., "--append"). Short-form options may be combined into a single argument (e.g., "-az" is equivalent to specifying both "-a" and "-z").

Any parameters to an option may be specified in the argument immediately following the option (e.g., "-c red" or "--color-scheme red"), or combined with the option into a single argument (e.g., "--color-scheme=red").

-a, --append
: Append the output to an existing PostScript file rather than overwriting it.

`-c`, `--color-scheme` ⟨*clr*⟩
: Use the color scheme ⟨*clr*⟩ for the output document. Currently defined colors include: `black`, `blue`, `green`, `red`, and `violet`.

`-d`, `--duplex`
: Arrange for the output to be printed duplex (on both sides of the paper) for printers which will honor the PostScript commands to do so.

`-h`, `--help`
: Print usage information and exit.

`-z`, `--compact-text`
: Compress large text blocks, such as spell descriptions, into space-saving dense blocks of text. Some typographical features (e.g., tables) are omitted entirely.

`-Z`, `--suppress-text`
: Go beyond `-z` and actually suppress large text blocks entirely.

SEE ALSO

`gma`(6), `gma-character-record`(5), `gma-printchar`(6).

AUTHOR

Steve Willoughby / steve@madscience.zone

HISTORY

This document describes version 4.1 of `printmon`.

gma-rendersizes(6)

NAME

gma rendersizes – prepare creature token images and other files for the GMA mapper

SYNOPSIS

```
gma rendersizes [-CdgikLlnSstw] [-M ⟨module⟩] [-a
⟨speed⟩[:⟨loops⟩]] [-c ⟨colors⟩] -f ⟨mapfile⟩ ⟨size⟩ ⟨imagefiles⟩...
gma rendersizes [-CdgiLlnSsw] [-M ⟨module⟩] [-a ⟨speed⟩[:⟨loops⟩]]
[-c ⟨colors⟩] -f ⟨mapfile⟩ grid|px ⟨width⟩ ⟨height⟩ ⟨imagefiles⟩...
gma rendersizes [-Cgm] prescale ⟨pixels⟩ ⟨squares⟩ ⟨imagefiles⟩...
gma rendersizes [-CdgiLlmnSsw] [-M ⟨module⟩] [-a
⟨speed⟩[:⟨loops⟩]] [-c ⟨colors⟩] -f ⟨mapfile⟩ scale ⟨pixels⟩ ⟨squares⟩
⟨imagefiles⟩...
gma rendersizes [-CdgiLlmnSsw] [-M ⟨module⟩] [-a
⟨speed⟩[:⟨loops⟩]] [-c ⟨colors⟩] -f ⟨mapfile⟩ postscale
⟨imagefiles⟩...
gma rendersizes [-iLlnw] [-M ⟨module⟩] -f ⟨mapfile⟩ map
⟨mapfiles⟩...
gma rendersizes [-CT] [-D ⟨density⟩] pdf ⟨files⟩...
```

DESCRIPTION

The gma-mapper(6) program supports image tiles as various on-map objects. These are used for random room features, floor tiles, and creature tokens.

Introducing tile images to the mapper is done by loading them in via a map file such as would contain other room features such as polygons, lines, etc. (See gma-mapper(5) for details.) The map file makes reference to the actual image files (which must be in GIF or PNG format due to implementation details of the Tk GUI toolkit, but rendersizes supports producing JPEG format as well) but does not include the image data inside the map file. These images will be loaded from local files or (more typically) over the network from a server and then cached locally for subsequent sessions.

Getting a set of images into the correct file format and naming conventions expected by the mapper is the job of the rendersizes program documented here. Since Tk also lacks default support for image scaling, the rendersizes program also pre-renders the image files at the correct scale for the five common zoom factors used during games (two up and down from the base view).

For creature token images, rendersizes scales the image for 11 sizes to allow it to be displayed at the five standard zoom levels but also to scale within each of those to different creature sizes.

Regardless of the location of the source image files, the resized image files are always written to the current directory.

In addition to the new image files, `rendersizes` will *append* image definition statements to the output file ⟨*mapfile*⟩ so that it contains the necessary references to the new image files. You can then load ⟨*mapfile*⟩ into the mapper to define all of the images for the map to use.

OPTIONS

`-a` ⟨*speed*⟩[`:`⟨*loops*⟩]
: If this option is given with one of the image output operations (`grid`, `px`, `scale`, `postscale`, and character token generation), it indicates that the GIF or PNG image data should be animated at the given ⟨*speed*⟩ (which indicates the number of milliseconds between each frame). The individual frames are stored in files with the prefix `:`⟨*n*⟩`:` prepended to its base filename, where ⟨*n*⟩ is the frame number starting with 0.

  Currently, only animated GIF files are supported for input. The output format should also be GIF (hence, we recommend use of the `-g` option as well).

  If ⟨*loops*⟩ is specified, clients should play the animation only that many times before stopping; otherwise they should play the animation continuously.

`-C`
: ImageMagick's `convert`(1) command was renamed to `magick` recently. Normally, `rendersizes` will prefer to invoke it as `magick` unless it can't find a command named that, in which case it will attempt to use the `convert` command instead. The `-C` option forces it to just assume `convert` is the command to use.

`-c` ⟨*colors*⟩
: In order to reduce the size of the files generated, you may add this option to indicate the number of colors which should be in the output image's palette. Typical usage is to specify 256 colors to get an 8-bit palette.

`-d`
: Limit output so that only zoom factor 2.0 files are generated (one step in from the default view).

`-D` ⟨*density*⟩
: Sets the dots-per-inch density for the operation (only applies to the `pdf` operation). Defaults to 300.

`-f` ⟨*mapfile*⟩
: Specifies the output map file in which to record the image information

for later loading into the mapper. If this is not specified, the image resizing will still occur but no map file output will be attempted.

N.B. In versions of GMA before 6.4, this was a positional parameter. It has now been moved to an optional one because it doesn't apply to all of `rendersize`'s operations.

`-g`

When creating local files (include prescaled images), `rendersizes` writes them in PNG format by default. With the `-g` option set, they will be written in GIF format instead. If the `-w` option is given, the files are written in GIF, PNG, and JPEG format anyway, so in that case `-g` has no additional effect.

`-i`

Generate filenames assuming a case-insensitive filesystem. This will try to avoid upper-case letters in generated names.

`-k`

Automatically include "killed" versions of each named creature token image. If a file is found in the same directory as the base token image which has a name appropriate to the killed version of that token, that file is used as-is. Otherwise, `rendersizes` will generate a new token image file in the same directory as the original which is a grayscale version of the original, and include that in the processed set of images. With this option selected, any explicitly-named input files which would be killed versions due to their name pattern are not processed.

`-L`

Deliver the output images and/or map files to your local mapper cache directory `~/.gma/mapper/cache`. This is analogous to the `-w` option which delivers them to a directory structure to be uploaded to a web server. Without this option, it would be necessary for you to use `-w` to generate the images, upload them to the server, then load the map file generated with the `-f` option into your mapper to have it download them to your mapper cache, before you could then work with them locally to prepare a map for your game. And that's a bit lame if you're still putting the maps together before it's ready to play. Using `-L` instead, you can simply load your cache with the images and immediately start working with them.

`-l`

By default, `rendersizes` writes the map file ⟨*mapfile*⟩ in the current map format. If the `-l` option is given, it will instead use a legacy format (version 17). Note that every time you run `rendersizes`, it appends to ⟨*mapfile*⟩, so you must be consistent. The file cannot contain a mixture of old- and new-format data.

`-M` ⟨*module*⟩
: Specify ⟨*module*⟩ as the module ID for the images being processed. This alters the opaque name encoding so that images with the same names as those used in other modules don't conflict.

  N.B. It is worth considering to make the names of image tiles unique to the module being played anyway, however, if the same group will be playing both modules at the same time. The ⟨*module*⟩ name given to this option will ensure that the filenames as stored server-side are different, but the image names used internally to `gma-mapper`(6) won't be. This means that a player's personal `mapper` client could confuse them if both are on screen at the same time, or if both were played close enough to each other that one module's images would still be in their local cache.

`-m`
: When using the `prescale` or `postscale` options, the default approach taken by `rendersizes` is to work with an initial prescaled file at the 1.0 zoom factor for the mapper. It will then be resized when you run `gma rendersizes postscale` to all the other sizes larger and smaller than 1.0. If you are starting from a high-resolution source image, however, better results may be obtained by specifying the `-m` option. This causes `gma rendersizes prescale` to first scale the image at the 4.0 zoom factor. Later, when running `gma rendersizes postscale`, it will only reduce the image to the other sizes. This avoids loss of resolution as would be the case with the default operation. If you use this option, you must specify it for *both* the `prescale` and `postscale` operations.

`-n`
: Write the map file in the new format (i.e., the current version supported by GMA Mapper), rather than a legacy format. This is the default behavior.

`-S`
: Limit the generated sizes to only those up to 2.0 zoom factor, inclusive.

`-s`
: Limit the generated sizes to only those up to 1.0 zoom factor, inclusive.

`-t`
: Automatically include semi-transparent versions of each named creature token image. If a file is found in the same directory as the base token image which has a name appropriate to the semi-transparent version of that token, that file is used as-is. Otherwise, `rendersizes` will generate a new token image file in the same directory as the original which is a semi-transparent version of the original, and include that in the processed set of images. With this option selected,

any explicitly-named input files which would be semi-transparent versions due to their name pattern are not processed.

`-T`

Directs ImageMagick to trim the image edges (using the `magick` program's `-trim` option) when processing PDF files.

`-w`

Instead of writing GIF files for local use, prepare a set of files to be served by a web server to the clients. This will create a version of each file in GIF, PNG, and JPEG format files, and assign them an ID code suitable for use on the web. These are written into a directory structure called `gma/map/...` which mirrors the directories on the web server, making it easier to copy that subdirectory tree to the web server.

CREATURE TOKENS

To prepare a tile image for a creature token, create it using any available image editing applications. Since creature tokens are circles, we recommend cropping the image with a circular cropping tool, leaving the area outside the circle as transparent. The `rendersizes` script assumes the *entire* image is intended to fit inside the boundaries of the map space for a creature of its size, and renders it in the square area of the appropriate number of grid squares. The transparent area around the circular cropping border allows the rest of the map outside the circular token to be visible (otherwise you'll end up with a square creature token). If any of the image inside the circle is transparent, the filled creature circle drawn by the mapper (white when the creature is alive and black if it's dead) will show through in those spaces.

Once a base image is prepared, it is given to the `rendersizes` script, which converts it to PNG format using ImageMagick, resizing it to occupy the correct space on the map. Five versions of the file are output, giving the correct size of image for the base zoom level, and two steps of in and out from there. (If the `-w` option is given, then 15 versions are output: five zoom levels for each of the three output formats.)

The ⟨*size*⟩ parameter indicates the size category for the creature using one of the single-character codes listed below under "Creature Sizes".

Naming Conventions

When a creature token is placed on the map, an image will be used if its name matches a certain convention based on the creature name on the map.

If a creature is named ⟨*name*⟩ (including instances called "⟨*name*⟩ #1", "⟨*name*⟩ #2", and so forth), then the mapper will use an image from the file "`#`⟨*name*⟩`@`⟨*pixels*⟩`.`⟨*type*⟩" where ⟨*pixels*⟩ is the length in pixels of each side of the image bounding box. If the creature is dead, then the image file "`%`⟨*name*⟩`@`⟨*pixels*⟩`.`⟨*type*⟩" is used instead. See the table below for the

Full	Shortened	Full	Shortened
`^0^`⟨*name*⟩`@`⟨*p*⟩`.`⟨*t*⟩	`#`⟨*name*⟩`@`⟨*p*⟩`.`⟨*t*⟩	`%^0^`⟨*name*⟩`@`⟨*p*⟩`.`⟨*t*⟩	`%#`⟨*name*⟩`@`⟨*p*⟩`.`⟨*t*⟩ or `%`⟨*name*⟩`@`⟨*p*⟩`.`⟨*t*⟩
`^1^`⟨*name*⟩`@`⟨*p*⟩`.`⟨*t*⟩	`=`⟨*name*⟩`@`⟨*p*⟩`.`⟨*t*⟩	`%^1^`⟨*name*⟩`@`⟨*p*⟩`.`⟨*t*⟩	`%=`⟨*name*⟩`@`⟨*p*⟩`.`⟨*t*⟩
`^2^`⟨*name*⟩`@`⟨*p*⟩`.`⟨*t*⟩	`-`⟨*name*⟩`@`⟨*p*⟩`.`⟨*t*⟩	`%^2^`⟨*name*⟩`@`⟨*p*⟩`.`⟨*t*⟩	`%-`⟨*name*⟩`@`⟨*p*⟩`.`⟨*t*⟩
`^3^`⟨*name*⟩`@`⟨*p*⟩`.`⟨*t*⟩	`+`⟨*name*⟩`@`⟨*p*⟩`.`⟨*t*⟩	`%^3^`⟨*name*⟩`@`⟨*p*⟩`.`⟨*t*⟩	`%+`⟨*name*⟩`@`⟨*p*⟩`.`⟨*t*⟩

Table D.1: Full and Shortened Token Image File Names

expected bounding box sizes. (The actual ID names used inside the mapper are different. This document only describes the operation of the `rendersizes` script which is looking at *file* naming conventions.)

Semi-transparent versions of token images have any of the allowed forms for living and killed creatures, prefixed by a "`!`" character (e.g., the semi-transparent version of "`#Bob`" could be "`!#Bob`").

Okay, that's not quite a complete description, but for the vast majority of creature tokens, the above description is all you need to consider, and is simpler than the full story. However, to accommodate the full set of use cases, including shapeshifters which may need multiple image files per token, as well as needing to have a consistent naming convention to make automation easier, the following general rules give the full naming convention.

A creature may have any number of appearances (called "skins"), numbered from 0 upward, with image 0 being the default appearance. The image used for a creature whose base name is ⟨*name*⟩ will be found in a file called "`^`⟨*skin*⟩`^`⟨*name*⟩`@`⟨*pixels*⟩`.`⟨*type*⟩", where ⟨*skin*⟩ is the skin number encoded as a decimal integer, and the other values are as described above. Likewise, the image used for the "dead" version of the creature will be "`%^`⟨*skin*⟩`^`⟨*name*⟩`@`⟨*pixels*⟩`.`⟨*type*⟩".

For compatibility with previous versions, and to allow a simpler naming convention, a few short-hand forms are recognized as shown in Table D.1

On input, only the initial part (up to the `@` sign) is expected. `rendersizes` will provide the rest automatically.

For example, you might prepare two image files at 500×500 pixel dimensions in PNG format for a creature called `Firefly`. The image to use while the firefly is alive will be saved as `#Firefly.png` and the one to use when it's dead will be `%Firefly.png`.

Assuming this is a medium-size creature, invoke `rendersizes` with the command

```
gma rendersizes -f __tiles__.map M /path/to/images/[#%]Firefly.png
```

This will cause `rendersizes` to convert these inputs into output files as shown in Table D.2.

Filename	**Description**
`#Firefly@50.png`	50×50 image for zoom 1.0 (alive)
`#Firefly@100.png`	100×100 image for zoom 2.0 (alive)
`#Firefly@200.png`	200×200 image for zoom 4.0 (alive)
`#Firefly@25.png`	25×25 image for zoom 0.5 (alive)
`#Firefly@12.5.png`	12×12 image for zoom 0.25 (alive)
`%Firefly@50.png`	50×50 image for zoom 1.0 (dead)
`%Firefly@100.png`	100×100 image for zoom 2.0 (dead)
`%Firefly@200.png`	200×200 image for zoom 4.0 (dead)
`%Firefly@25.png`	25×25 image for zoom 0.5 (dead)
`%Firefly@12.5.png`	12×12 image for zoom 0.25 (dead)

Table D.2: Output Files For Firefly Token Image.

	Pixel Size by Zoom Step				
Size Code	**–2**	**–1**	**±0**	**+1**	**+2**
`F`, `f`	1	2	5	10	20
`D`, `d`	2	5	10	20	40
`T`, `t`	6	12	25	50	100
`1`, `S`, `s`, `M`, `m`	12	25	50	100	200
`2`, `L`, `l`	25	50	100	200	400
`3`, `H`, `h`	37	75	150	300	600
`4`, `G`, `g`	50	100	200	400	800
`6`, `C`, `c`	75	150	300	600	1200

Table D.3: Token Images and Creature Size Categories

Creature Sizes

The ⟨*size*⟩ parameter specifies the token size in game terms, using the same size codes which are recognized by the mapper, which is translated to the map pixel sizes as shown in Table D.3. A pixel size of ⟨n⟩ means the image is sized to fit in a square box of ⟨n⟩×⟨n⟩ pixels.

ARBITRARY IMAGE TILES

Tiles intended for other uses on the mapper can be of any arbitrary rectangular dimensions. If the space on the map is not perfectly rectangular, use some transparent areas around the visible image parts to let overlapped areas show through, but the image itself must be a rectangle.

Prepare the image as described above for character tokens, but unless the image file fits in the sizes used for character tokens, you will need to give explicit image dimensions. Instead of giving the single ⟨*size*⟩ param-

eter used for creature tokens, give the keyword `grid` or `px` followed by two integer values ⟨*width*⟩ and ⟨*height*⟩. Depending on the keyword used, these specify the image ⟨*width*⟩ and ⟨*height*⟩ in units of grid squares or pixels, respectively. The source images will be resized to fit inside this maximum bounding rectangle while maintaining their original aspect ratio.

Naming Conventions

The images can be called anything not already reserved for use as a creature token (essentially this means that the file should not start with ":", "~", "#", "%", "!", "=", "-", or "+", but to be safe we recommend starting the name with a letter). A good convention to use would be ⟨*module*⟩`_`⟨*name*⟩ where ⟨*module*⟩ is some unique identifier for the adventure module the image is designed for or a general indicator of a module-independent type of image.

On output, the files generated for an input image called "⟨*name*⟩" will be "⟨*name*⟩`@`⟨*wp*⟩`x`⟨*hp*⟩`.`⟨*type*⟩" where ⟨*wp*⟩ and ⟨*hp*⟩ are the scaled width and height in pixels, and ⟨*type*⟩ is the file type.

BACKGROUND IMAGES

Images intended to be a bitmap that underlies the floorplan of one or more rooms need to be scaled to the mapper program's graphic scale so that everything lines up where they should. This is done by specifying one or more such images using the `scale` keyword rather than `grid` or `px` as described above. In this case, the `scale` keyword is followed by two parameters: ⟨*pixels*⟩ and ⟨*squares*⟩, which together specify how many pixels in the image represents the given number of five-foot squares on the map. This will produce a map file that can be placed as any other tile, but will be the correct size to use as the floor of the room.

Oddly-shaped rooms (i.e., those which are not rectangles with sides parallel to the *x* and *y* axes) will probably need to have transparent bits cut out where they would overlap with adjacent rooms.

These files should follow the same naming convention as image tiles (since that's what they are).

Sometimes you will want to scale a starting image first to match the grid size, then make a number of edits (perhaps slicing the image into individual areas you want to display separately) before rendering it into all the final sizes and formats.

In this case, start with the `prescale` mode just as you would with the `scale` mode explained above. This will not write anything to ⟨*mapfile*⟩, since it's not making the final image(s) yet. Instead, it makes the initial scale for zoom level 1.0 which is written into an output file called `pre_`⟨*imagefile*⟩`.png` (or `pre_`⟨*imagefile*⟩`.gif` if the `-g` option was given, unless you are also using the `-w` option, but that option is not recomended with `prescale`).

Once your edits are made, run the final files through `rendersizes` using the `postscale` mode to generate the final set of images.

A good way to get a high-resolution image file to work with from a PDF document to use the `pdf` operation, specifying `-D` and/or `-T` options as needed.

MAP FILES

If the keyword `map` is given for the ⟨*size*⟩ parameter, then the script expects the filename(s) to be map files rather than images. These are renamed and stored for upload to the server. (This implies the `-w` option as well, whether or not it is explicitly given on the command line.)

EXAMPLES

Two PNG images files are created as creature tokens for Alice and Bob. These are placed in files named `#Alice.png`, and `#Bob.png`. These characters are medium-size, so we prepare them for use with the mapper by running:

```
$ gma rendersizes -k -f __tiles__.map M ./#{Alice,Bob}.png
Converting images to M-size image tiles into __tiles__.map...
#Alice.png: 12 25 50 100 150 200 300 400 600 800 1600 (existi
ng)
%Alice.png: 12 25 50 100 150 200 300 400 600 800 1600 (genera
ted)
#Bob.png: 12 25 50 100 150 200 300 400 600 800 1600 (existing
)
%Bob.png: 12 25 50 100 150 200 300 400 600 800 1600 (generate
d)
```

When it completes, a collection of PNG image files are created: `#Alice@12.png`, `#Alice@25.png`, and so on.

Additionally, the `__tiles__.map` file has descriptions of Alice and Bob's image files added to it.

Now when running the mapper, the GM needs only to load `__tiles__.map` first (just like loading any map file) to have all those images available for creature tokens.

Next, a floor tile covering room A12 in module M1 is created in a source file called `m1_a12.png` which has the correct aspect ratio for the room. On the map, it occupies 3 map squares across and 4 squares down. This is prepared for a mapper connected to an internet server by running:

```
$ gma rendersizes -w -M M1 -f __tiles__.map grid 3 4 \
m1_a12.png
Converting images to 3x4-size image tiles into __tiles__.map.
..
m1_a12.png: 37x50 75x100 150x200 300x400 600x800
```

This will create 15 output files representing the room a12 in five different zoom factors, such as `7SCpWdQQzNhD2eHfzuBCKAm1_a12@37x50.gif`, `7SCpWdQQzNhD2eHfzuBCKAm1_a12@75x100.gif`, `7SCpWdQQzNhD2eHfzuBCKAm1_a12@150x200.gif`, `7SCpWdQQzNhD2eHfzuBCKAm1_a12@300x400.gif`, and `7SCpWdQQzNhD2eHfzuBCKAm1_a12@600x800.gif`, and correspondingly for PNG and JPEG file formats. (Note that the file names have a prefix added in front of the base "`m1_a12`" name which is expected for files stored on external servers. This is done to obfuscate the names to discourage someone from getting spoilers by enumerating the image files.)

To store a set of map files on the server for faster retrieval by clients later:

```
$ gma rendersizes -w -M M1 -f __tiles__.map map S0.map S1.map
Converting map files to __tiles__.map...
S0.map: S0.map -> gma/map/_/_k/_kNzAwcJiBGX-WBw6uy2Hw.map
S1.map: S1.map -> gma/map/Z/Z-/Z-YnTgrAvYkvmx7AmiJT-A.map
```

SEE ALSO

ImageMagick documentation, `gma-mapper`(5), `gma-mapper`(6).

BUGS

Support for animated images is still somewhat experimental. A few caveats:

1. If a source image is an animated GIF or PNG file, you must use the `-a` option to treat it as an animated file and extract the frames to individual files. If you don't, ImageMagick will inadvertently end up breaking it up into frames anyway when converting to other formats, but this will be unexpected by the rest of the system and won't properly work.

2. You should use `-g` if the source image was in GIF format (i.e., you need to produce the same output file format as the input was). Note that the `-w` option will still generate all the frame files in all the usual formats from there.

3. Automatic generation of "killed" and semi-transparent creature tokens does not yet work for animated creature token images. You will

need to prepare separate animated or static image files for those tokens.

gma-roll(6)

NAME

gma roll – roll RPG dice

SYNOPSIS

```
gma roll [⟨diespec⟩...]
```

DESCRIPTION

This utility makes the GMA toolkit's built-in die-rolling facility available on the command line to make it easy to make die rolls independent of running the rest of GMA.

One or more ⟨*diespec*⟩ expressions may be given on the command line, in which case each of them is rolled (in the order given) and the results of those rolls printed to the standard output.

If none are specified on the command line, `roll` goes into interactive mode and prompts for a ⟨*diespec*⟩ to be entered at the terminal, reports the results, and prompts for another. Entering "`exit`", "`quit`", or "`q`" will terminate this input loop and exit the `roll` program.

A ⟨*diespec*⟩ specifies a die roll to be made. See `gma-dice-syntax`(7) for a full description of what you can put in this specification.

Whitespace inside the ⟨*diespec*⟩ expression is ignored. Specifying "12 d 6 +2" is the same as "12d6+2".

SEE ALSO

`gma-dice`(3), `gma`(6). `gma-go-roll`(6), `gma-dice-syntax`(7).

AUTHOR

Steve Willoughby / steve@madscience.zone

HISTORY

This document describes version 4.1 of `roll`. A version of `roll` was also in version 3 of GMA, but was different in operation.

gma-rotate(6)

NAME

gma rotate – rotate image sets for use by GMA

SYNOPSIS

`gma rotate [-h] [-d ⟨degrees⟩[:⟨degrees⟩...]] ⟨file⟩ ...`

DESCRIPTION

Occasionally, when preparing images for use with the mapper, you need to provide a number of variations of an image tile at different rotations. To easily accomplish this, run `gma rotate` first to produce the desired rotations, then process all the resulting files through `gma rendersizes` as normal.

The output files will have the same name as the original, with "_⟨*degrees*⟩" appended before the file extension. Given this file naming convention, any input file which already has a suffix like that in its name will not be processed by `gma rotate` on the assumption that it's already a rotated version of another file, and we don't want to rotate it again.

OPTIONS

`-d, --degrees` ⟨*degrees*⟩`[:`⟨*degrees*⟩`...]`
: Specify a colon-separated list of rotation angles for the image. An output file will be produced for each of these as described above. Defaults to `90:180:270` if not specified.

`-h, --help`
: Print usage information and exit.

SEE ALSO

`gma-rendersizes`(6).

gma-servercheck(6)

NAME

gma servercheck – Check health of running GMA server

SYNOPSIS

```
gma servercheck -h
gma servercheck --help
gma servercheck [-Dq] [-C ⟨file⟩] [-H ⟨host⟩] [-P ⟨protocol⟩] [-p ⟨port⟩]
gma servercheck [--config ⟨file⟩] [--debug] [--host ⟨host⟩] [--port
⟨port⟩] [--protocol ⟨protocol⟩] [--quiet]
```

DESCRIPTION

This utility connects to a running GMA gma-go-server(6) program to check that it's still running and does not appear to be deadlocked. In normal operation, it will print out the health stats and uptime to the standard output.

OPTIONS

The following command-line options control the program's behavior:

`-C, --config` ⟨*file*⟩
: Read from the configuration ⟨*file*⟩ to set options in addition to, or instead of, command-line options, in the same fashion as the gma-mapper(6) client.

`-D, --debug`
: Print additional information to stdout which traces through the program's operation and data received from the server.

`-H, --host` ⟨*hostname*⟩
: Specify the name of the GMA server.

`-h, --help`
: Print usage information and exit.

`-p, --port` ⟨*port*⟩
: Specify the TCP port number of the server.

`-P, --protocol` ⟨*protocol*⟩
: Normally, it will be reported as an error if the server's implemented protocol is not the same as the current protocol that goes with this version of servercheck. This option allows you to specify a different

protocol number you expect the server to be using. If you specify a hyphen ("-") in place of ⟨*protocol*⟩, no check will be made at all.

`-q, --quiet`
: Suppress the normal output printed by the program.

RETURN VALUE

The following return values are sent to the shell:

0 The server is operating as expected.

1 The server's state could not be determined.

2 The server's main event loop hasn't reported by the expected time, meaning that the server may be deadlocked or otherwise compromised.

3 The server's protocol version is not what was expected.

DEPENDENCY

This program requires the third-party `dateutil` package. If you don't already have it, you can install it by running:

```
pip install python-dateutil
```

AUTHOR

Steve Willoughby / steve@madscience.zone.

gma-session-publish(6)

NAME

gma session-publish – redact and publish game session transcripts for public consumption

SYNOPSIS

`gma session-publish --help`
`gma session-publish -s -j` ⟨*jsonfile*⟩ [`-F` ⟨*htmlfile*⟩] [`-S` ⟨*htmlfile*⟩]
`gma session-publish --summary` —-playtimes ⟨*jsonfile*⟩ [`--footer` ⟨*htmlfile*⟩] [`--synopsis-header` ⟨*htmlfile*⟩]

`gma session-publish` [`-c` ⟨*name*⟩] [`-F` ⟨*htmlfile*⟩] [`-H` ⟨*htmlfile*⟩] `-j` ⟨*jsonfile*⟩ [`-L` ⟨*linkfile*⟩] [`-l` ⟨*linkfile*⟩] [`-p` ⟨*popupfile*⟩] [`-t` ⟨*title*⟩] ⟨*input-files*⟩...

`gma session-publish` [`--campaign` ⟨*name*⟩] [`--footer` ⟨*htmlfile*⟩] [`--html-header` ⟨*htmlfile*⟩] [`--link-filter` ⟨*linkfile*⟩] [`--links` ⟨*linkfile*⟩] `--playtimes` ⟨*jsonfile*⟩ [`--popup` ⟨*popupfile*⟩] [`--title` ⟨*title*⟩] ⟨*input-files*⟩...

DESCRIPTION

This is a small helper script used for the author's game group but may be useful to others so is offered along with the rest of the tools just in case.

In our group, we use a typical online web forum to collaborate with each other. Part of that includes a space where we write up a summary of what happened during each game session. Being me, I like to do that with a certain amount of style, formatting it nicely with some included maps and other illustrations, HTML display elements that hide the blow-by-blow combat transcripts until you want to actually see them, etc. I prepare these with the help of GMA's `gma-forumpost`(6) script to make it quick and easy to put together a summary for my players to refer to. This includes some links to our private video recorting of the game session at the start of each combat and at other points of interest.

All of this is stored on our private forum where only we can see it. However, we have a few friends who have expressed interest in seeing the transcript of our games, but who don't have access to our private forum. We also have not allowed access to the videos outside our group, since we're not performing to an audience, just making a recording of our games for our own reference, and Internet safety and privacy are real things too.

This script is thus employed to take the prepared forum posting HTML files (after being processed by the above), removes the references to the

videos and other things internal to our web forum, and produces a standalone version of the game transcripts which can be viewed by whomever.

OPTIONS AND PARAMETERS

Normally, the script formats a set of pages that go together to give the gameplay summaries for the game sessions for a single "book" of a multi-part adventure path. All of the ⟨*source-files*⟩ named on the input are assumed to be HTML source files whose names follow a certain convention:

⟨*book*⟩.⟨*session*⟩[⟨*extra*⟩].`html`

where ⟨*book*⟩ and ⟨*session*⟩ are integers which give, respectively, the book number in the adventure path series we're playing and the game session number in the AP. Both start at 1 and count upward incrementing by 1 without starting over. If there are multiple files that together comprise a single session's transcript, then the optional ⟨*extra*⟩ field is used, which must consist entirely of lower-case letters and by convention starts at `a` and "counts" up from there in a way that the file namess will naturally sort in the correct order.

Thus, the first game session would be recorded in the file `1.1.html`, and the next in `1.2.html`. If the third session had two supplimental postings that added material, all of which belong to that session, the three files would be named `1.3.html`, `1.3a.html`, and `1.3b.html`. If we start Book 2 during session #7, that game would be recorded in file `2.7.html`. Note that the "sort order" of the ⟨*extra*⟩ field is a convention only; the files will be processed in the order presented on the command line, but the semantic meaning of the ⟨*book*⟩ and ⟨*session*⟩ fields is fixed.

The ⟨*page-title*⟩ parameter is set in the HTML document title tag to be displayed in the browser.

The script otherwise takes the followng options:

`--campaign`/`-c` ⟨*name*⟩
: Sets the name of the campaign as it will appear on the web pages. This should look reasonable with the word "Campaign" following it.

`--footer`/`-F` ⟨*htmlfile*⟩
: Specifies the HTML footer file to include at the bottom of all output, just before the `</BODY></HTML>` tags. If not given, defaults to `~/.gma/session-footer.html`.

`--html-header`/`-H` ⟨*htmlfile*⟩
: Specifies an HTML header file to include at the start of each regular page, after sending the initial `<HTML>` tags. Any occurrence of the string `{title}` will be replaced with the value of the `--title` option.

`--link-filter`/`-L` ⟨*linkfile*⟩
: Image files usually need their source URLs translated because the location of internal images may be different from where things are re-

hosted for public viewing. This file provides a number of substitutions to make for `<IMG>` tags. Each line has the form

/⟨*regex*⟩/⟨*replacement*⟩/

Unlike the other similar files, these are regular expression matches, not plain literal string matches. Any `<IMG>` tag whose `src` attribute matches the regular expression ⟨*regex*⟩ is changed so that attribute instead refers to ⟨*replacement*⟩.

`meta link` tags should also be implemented in local header files and then any reference to them in the original HTML files deleted. To accomplish this, any `link` URLs which match ⟨*regex*⟩ and whose ⟨*replacement*⟩ string is empty are dropped from the output. Any remaining `link` URLs cause an error since the point is to remove references to internal links as part of this process. If not given, defaults to `~/.gma/session-link.filter`.

`--links/-l` ⟨*linkfile*⟩

Any hyperlinks which are referenced in the original forum posts need to be scrubbed or altered for the public version of the game summaries. This file provides a set of substitutions to apply to any links found, one per line. Each line has the format

/⟨*orig*⟩/⟨*replacement*⟩/

Any hyperlink whose `href` attribute exactly matches the string ⟨*orig*⟩ is changed to instead refer to ⟨*replacement*⟩. The delimiter character shown here as "/" can be any single character not otherwise appearing in either string. If this option is not given, it defaults to `~/.gma/session-publish.links`.

`--playtimes/-j` ⟨*jsonfile*⟩

Specify the data file which describes all the game sessions that have taken place. This JSON format file must contain an object with an attribute called `game_sessions`. This in turn is a list of objects describing each game session, each of which has at least the following attributes:

`book`
: The book number within the campaign where this game session takes place, as an integer.

`date`
: The date in the real world when the game session was played.

`title`
: The title you wish to give to the game session.

`world_dates`
: The date(s) in the game world covered by the game session.

`synopsis`
: A brief synopsis of the main events of the game session. These will be placed on the summary page.

The text in these fields is written to the HTML output just as they are in the file, so any HTML or JavaScript that appears here is preserved intact without escaping or translation.

`--popup/-p` ⟨*popupfile*⟩
: Links to images in the original, to show illustrations relevant to the campaign session, will be replaced with pop-up images when moused over thanks to the definitions stored in this file. Each line in ⟨*popupfile*⟩ describes one popup image to replace a hyperlink from the original HTML page. These lines each have the form:

/⟨*ID*⟩/⟨*title*⟩/⟨*orig*⟩/⟨*replacement*⟩/

Any hyperlink whose `href` attribute exactly matches the string ⟨*orig*⟩ is changed to instead refer to ⟨*replacement*⟩, using a browser pop-up with unique ID ⟨*ID*⟩ and title string ⟨*title*⟩. The delimiter character shown here as "/" can be any single character not otherwise appearing in either string. As a special case, if ⟨*replacement*⟩ has the form "<<⟨*name*⟩", then rather than linking to an external file, the following lines of the file, up to a line containing only the text ⟨*name*⟩, are the HTML content to pop-up in the browser. If this option is not given, it defaults to `~/.gma/session-publish.links`.

`--summary/-s`
: Normally, the script gathers all the ⟨*input-files*⟩ listed on the command line and emits to its standard output a single HTML document built from those files which describes the game sessions for that book. With this option, no input files are read. Instead, a summary page is generated, built entirely from the data in the `playtimes.json` file.

`--synopsis-header/-S` ⟨*htmlfile*⟩
: Specifies an HTML file to include at the start of the summary page (after sending `<HTML>`). If not given, this defaults to `~/.gma/session-synopsis.html`.

`--title/-t` ⟨*title*⟩
: Sets the title of the page, which is generally a book from the campaign.

TYPICAL SETUP

Translating a website has a lot of little pieces to translate and adjust along the way, which is why this whole script is set up the way it is. It's a little wonky and fragile but once set up, it runs fairly smoothly from there on. A typical setup is to write a copy of your internal game summary HTML files in a `src` subdirectory which also includes a `Makefile` to build

the public website from them. The structure of the whole public site is thus:

`0.html`	Main homepage for the campaign story, static handwritten page.
`1.html`	Book I game summary page created by script.
`1.html`	Book II game summary page created by script.
`tldr.html`	Game synopsis page created by script `--summary` option.
`gamers.css`	Local CSS style definitions.
`src/`	Source directory with original forum data files and build support.
`src/Makefile`	Build automation instructions.
`src/1.0.html`	Base Book I page (top, before transcripts start, handwritten HTML).
`src/1.1.html`	Book I, session 1 game transcript original HTML file.
`src/1.2.html`	Book I, session 2 game transcript original HTML file.
`src/1.3.html`	Book I, session 3 game transcript original HTML file.
`src/1.4.html`	Book I, session 4 game transcript original HTML file.
`src/2.0.html`	Base Book II page (top, before transcripts start, handwritten HTML).
`src/2.5.html`	Book II, session 5 game transcript original HTML file.
`src/2.5a.html`	Book II, session 5 game transcript (addendum) original HTML file.
`src/2.6.html`	Book II, session 6 game transcript original HTML file.
`src/footer.html`	HTML footer for all output files.
`src/gamers.popups`	Definitions for pop-up images.
`src/gamers.link-filter`	Flters for image source URLs.
`src/gamers.links`	Hyperlink substitution file.
`src/header.html`	HTML header for game transcript files.
`src/playtimes.json`	Game session data.
`src/synopsis.html`	HTML header for synopsis (tldr) file.

SUPPORTING FILES

Makefile

The `Makefile` that governs how to build all the public files in this case would look like this:

```
BOOK1SRC=1.0.html 1.1.html 1.2.html 1.3.html 1.4.html
BOOK2SRC=2.0.html 2.5.html 2.5a.html 2.6.html
JSON=playtimes.json
CFG=--playtimes $(JSON) --campaign "Amazing Adventure" \
  --footer footer.html --html-header header.html \
  --link-filter gamers.link-filter --links gamers.links \
  --popup gamers.popups
all: ../tldr.html ../1.html ../2.html

../1.html: $(BOOK1SRC) $(JSON)
  gma session-publish $(CFG) --title "Book I" $(BOOK1SRC) \
   > ../1.html
```

```
../2.html: $(BOOK2SRC) $(JSON)
  gma session-publish $(CFG) --title "Book II" $(BOOK2SRC) \
   > ../2.html

../tldr.html: $(JSON)
  gma session-publish $(CFG) --summary --synopsis-header \
   synopsis.html > ../tldr.html
```

0.html

The starting page for the game summaries is not built or maintained by the script, but is just a static HTML file you set up to read appropriately to link to the rest. In our example here we call it `0.html` but it could be anything. It can be something like this:

```
<html>
 <head>
  <title>Our Amazing Game :: Amazing Adventure Campaign :: Game
Summaries :: Index<title>
  <link rel="stylesheet" type="text/css" href="/site.css" />
  <link rel="stylesheet" type="text/css" href="gamers.css" />
  <meta charset="UTF-8" />
 </head>
 <body>
  <h1>Amazing Adventure</h1>
  <p>Welcome to our game!</p>
  <p>You can read <a href="tldr.html">a very condensed plot
synopsis here</a> if you don't want to curl up with your
web-enabled tablet next to a warm fire for an evening of
reading through the complete saga in the next section.</p>
  <ul>
   <li><a href="1.html">Book I: <em>Dinner with the Goblin Clan
</em></a></li>
   <li><a href="2.html">Book II: <em>Zombie Invasion!</em></a>
</li>
   <li><span style="color: #bbbbbb">Book III</span></li>
   <li><span style="color: #bbbbbb">Book IV</span></li>
  </ul>
 </body>
</html>
```

Note that this links to the CSS files in play and to the other generated content.

CSS

The CSS file (`gamers.css` in our example) you set up should define any styles that will be used by your HTML input. For example:

```
table.pftable {
 border-collapse: collapse;
}
table.pftable td {
 padding-left: 5px;
 padding-right: 5px;
}
table.pftable tbody tr:nth-child(odd) {
 background-color: rgba(0,0,0,.15);
}
table.pftable th {
 font-weight: bold;
}
.module_excerpt:hover {
 border-style: dashed;
 border-color: #0000ff;
 border-width: 1.5px;
 padding: 1px;
 display: inline-block;
}
```

x.0.html

In the source directory, the start of each book has a static portion in a file called ⟨*x*⟩`.0.html` where ⟨*x*⟩ is the book number. This provides whatever fancy header and introduction text you'd like to start with before the generated content is added from the game session files. For example:

```
<h1>Amazing Adventure Campaign</h1>
<h2>Book I: <em>Dinner with the Goblin Clan</em></h2>
<p>Here's where our story begins...</p>
```

HTML Header

This file contains the header that is inserted at the top of all regular HTML files between the `<HTML>` tag and the content. For example:

```
```

```
 <head>
  <title>Amazing Adventure Campaign :: Game Summaries :: {title}
 </title>
  <link rel="stylesheet" type="text/css" href="/site.css" />
  <link rel="stylesheet" type="text/css" href="gamers.css" />
  <meta charset="UTF-8" />
 </head>
 <body>
  <div class="content">
```

Synopsis Header

Same as above, but for the synopsis page only.

Footer

Any last bits of HTML such as a copyright notice or whatever you want added to the bottom of all HTML files is included in this file.

BUGS

This is a very simple, bespoke script that just does one particular function and wasn't designed to be too much of a general-purpose tool, so it's likely to have a number of rough edges that wouldn't have been there if it had been designed more purposefully from the beginning. Perhaps one day it'll get refactored.

AUTHOR

Steve Willoughby / steve@madscience.zone.

gma-show(6)

NAME

gma show – Show various GMA campaign world values

SYNOPSIS

```
gma show [-cdFhimSsw] [-f ⟨regex⟩]
gma show [--help]

gma show [--classes] [--detail] [--feats] [--filter ⟨regex⟩]
[--monsters] [--no-case] [--skills] [--spells] [--weapons]
```

DESCRIPTION

This utility allows you to easily see core campaign data as defined in your GMA installation.

If given one or more options that specify types of data such as skills, `show` will print the list of defined skills with the codes by which they are known in GMA (which may be necessary to specify in GMA's data files).

More detail may be requested by giving the options documented below.

OPTIONS

The following options control the behavior of `show`.

`-c`, `--classes`
: Include defined character classes in the output.

`-d`, `--detail`
: Include full details about the selected items.

`-F`, `--feats`
: Include defined feats in the output.

`-f`, `--filter=`⟨*regex*⟩
: Include only those items whose codes match the regular expression ⟨*regex*⟩.

`-h`, `--help`
: Print usage information and exit.

`-i`, `--no-case`
: Make the pattern match used by `--filter` disregard case.

`-m`, `--monsters`
: Include defined monsters in the output.

`-S`, `--spells`
: Include defined spells in the output.

`-s`, `--skills`
: Include defined skills in the output.

`-w`, `--weapons`
: Include defined weapons in the output.

SEE ALSO

`gma`(6).

HISTORY

`show` first appeared in version 6.2 of GMA Core.

AUTHOR

Steve Willoughby / steve@madscience.zone

BUGS

Apologies to fuctional programming fans for the name of this script.

gma-skillranks(6)

NAME

gma skillranks – generate player skills reports

SYNOPSIS

```
gma skillranks [-hpH] [-c ⟨file⟩] [-m ⟨file⟩] [-P ⟨file⟩] [-w ⟨name⟩]
gma skillranks [--conf ⟨file⟩] [--help] [--html] [--matrix ⟨file⟩]
[--playerreport] [--preferences ⟨file⟩] [--prune] [--world ⟨name⟩]
```

DESCRIPTION

Takes the character information from the character files named in the configuration file and tabulates combat statistics, attributes and skill ratings, and then generates an output summarizing them in a form suitable for player reference or for GM reference, as documented below.

OPTIONS

`-c`, `--conf` ⟨*file*⟩
: Read the configuration information from ⟨*file*⟩ (as described below). If this is not given, it defaults to `/opt/gma/state.ini` for...reasons.

`-H`, `--html`
: Render the player report in HTML rather than plain text.

`-h`, `--help`
: Print a short summary of the usage and options, then exit.

`-m`, `--matrix` ⟨*file*⟩
: Generate a GM reference matrix into the named ⟨*file*⟩. The output is in PostScript format. It summarizes information about the entire party in one sheet, for running combat encounters. Includes hit points, armor class, attack modifiers, ability scores, skills, etc.

`-p`, `--playerreport`
: Generate a reference sheet for the players' use. It lists each of several skills and gives a list of character names for each, in order starting with the character best at that skill to the one least capable with it. Significant gaps between one character's ability and the next on the list are indicated with slashes (/). This reflects the general idea of player skills which the party would have noticed from their association, without breaking the fourth wall to compare plusses on character sheets. This is output to the standard output device.

`-P`, `--preferences` ⟨*file*⟩
: When using `skillranks` to update GMA's party stats, this option specifies the ⟨*file*⟩ name of the GMA preferences file to use. The default is `~/.gma/preferences.json`.

`--prune`
: When using `--world` to update campaign data, remove any characters who are not mentioned in the collected data. In other words, this replaces all campaign party data with the data collected by `skillranks` instead of just updating them.

`-w`, `--world` ⟨*name*⟩
: When using `skillranks` to update GMA's party stats, this option specifies the name of the campaign world containing the party to be updated from the data in the character files.

CONFIGURATION FILE

The configuration file is in the usual INI format used by the rest of the GMA tool suite. There may be other game status information in that file used by other tools, but the `skillranks` program looks for a `characters` field in the `[files]` stanza and a `members` field in the `[party]` stanza.

The `characters` field gives the name of the directory in which the GCR files are available. This is a multi-line field. The directory named on each line of this field will be searched in the order listed until a character file with the expected name is found.

The `members` field lists each of the character files (excluding the `.gcr` suffix) for the party's characters, separated by white space.

The `[party]` stanza may also contain a `notes` field in the same format as the `notes` field in character record files. This adds a two-column notes section to the lower half of the page. The `<<-->>` marker starts the next column. (Currently this new feature needs a bit more work since it doesn't yet correctly handle reaching the end of the page with a long notes section.)

Ad-hoc Character Stats

If you have characters managed outside GMA (and thus, have no `.gcr` file) that you want to include in the report, you can include their essential stats in a stanza with the name `[char.`⟨*id*⟩`]` where ⟨*id*⟩ is the character ID code as listed in the `[party] members` field described above. In this stanza you can include the fields `name`, `classes`, `levels`, `race`, `align`, and `hp`, just as they would appear in the character `.gcr` file, and the fields `fort`, `refl`, `will`, `skills`, `bab`, `cmb`, and `cmd`, as they would appear for monsters in a `.gmo` file or animal section of a `.gcr` file.

Ability scores are entered in fields called `str`, `dex`, `con`, `int`, `wis`, and `cha`, where a plain integer value (e.g., "`12`") is taken to be a raw ability score but

one beginning with a + or - character (e.g., "+1") is taken to be an ability score modifier.

For the ability scores and saving throws, entering "--" or "---" indicates that no value exists at all.

Initiative modifiers are placed in a field called `init_add`, and includes the sum of all the modifiers to initiative *except* from Dexterity.

Armor class values (just the final totals, not the components which comprise them) are entered in fields `ac`, `touch`, and `flat`.

The list of spell DCs for the character are listed in a slash-separated value in the field `dc`.

Most fields have reasonable defaults. Ability scores which are omitted are undefined (as in, the character has no ability score at all). The `name`, `classes`, and `levels` fields are required.

The fields given in the `[char.`⟨*id*⟩`]` stanzas are not checked or processed as the `.gcr` files are; they are taken and printed verbatim as given in this file. This is a mixed blessing—it means you can take advantage of this to put the data you want and need on the matrix sheet, but it also means GMA isn't looking over your shoulder to make sure the values conform to the game rules at all.

Using GCR and POR Character Files With Ad-hoc Updates

If you have both a `.gcr` file (or a Hero Lab `.por` file) and a `[char.`⟨*id*⟩`]` stanza, then the information will be read from the `.gcr` file first, then all the fields found in the `[char.`⟨*id*⟩`]` stanza will override what was in the file.

Updating GMA Campaign Party Information

Another mode of operation for `skillranks`, initiated by the inclusion of the `--world` (or `-w`) option, is to copy the collected character stats into the specified campaign world data in the GMA preferences file.

If there are any characters in the GMA preferences file which do not appear in the data collected by `skillranks`, they will be left unaltered unless the `--prune` option is also included.

This operation requires data fields not normally used by the other modes. You may need to provide these in a `[char.`⟨*id*⟩`]` stanza.

`init_dice`
: The die-roll specification to use when rolling initiative for this character. Defaults to "`d20`" but can be changed to, for example, "`d20 best of 2`" if a character gets to reroll initiative and take the better of the two rolls.

`is_pc`
: This boolean value is true by default. If true, it marks the character as a PC in the party. If false, they are an NPC that is being tracked in the campaign but not one of the PCs.

`blur_hp`
: The percentage by which to blur hit points reported to the mapper UI for this character. The default, 0, means to report hit points accurately. Valid values range from 0 to 100.

`caster_level`
: This is a multi-line value. Each line provides the character's caster level in the form ⟨*class*⟩ | ⟨*CL*⟩ to enable you to specify multiple different caster levels if the character has multiple spellcasting abilities at different effective levels. Specifying "`*`" as the ⟨*class*⟩ provides the overall (or only) caster level for single-class casters.

Note that if you provide modifiers (as opposed to raw ability scores) for Dexterity and Constitution, the minimum raw score which provides that modifier will be set in the campaign data for that character. If you specified raw ability scores, then those raw values will be used.

Importing From Hero Lab Portforlios

If no `.gcr` file is found for a character, but a `.por` file is, `skillranks` will attempt to import the data it needs from that Hero Lab portfolio file. This is an experimental feature and not guaranteed to be completely functional at this point. You may need to compensate for its shortcomings by adding data in a `[char.`⟨*id*⟩`]` stanza as well.

The data pulled from the `.por` file is taken from the statblocks data saved there (preferring the XML version if one is found, falling back to the plain-text one if necessary).

Meta Configuration

The configuration file may contain some metagame settings in the `[meta]` stanza. The following field may be defined there:

`spelling_corrections`
: This is a multi-line list. Each line (which must be indented after the first line to indicate that the indented lines all belong to the same field) specifies a correction to the spelling of a name as taken from a character file, to correct for mistakes or to reassign codes where there is a mismatch between, for example, Hero Lab and GMA.

  Each line is of the form "⟨*old*⟩ => ⟨*new*⟩" where spaces at the start and end of the line and around the => symbol are ignored. Values named ⟨*old*⟩ will be changed to ⟨*new*⟩ as they are read in from the data files. Currently, this is defined only for skill names, and is substituted after translating the skill name to GMA form (e.g., "`knowedge.arcana`"). Thus, if a Hero Lab file had been set up with a skill called "Perform (Cinema)" but the skill as known to GMA is actually "`perform.stage`"

then the line "`perform.cinema=>perform.stage`" should be added to this field.

SEE ALSO

`gma-character-record(5)`, `gma(6)`.

AUTHOR

Steve Willoughby / steve@madscience.zone

HISTORY

Campaign data and updated data handling were introduced with GMA 6.23. Importing character data from Hero Lab `.por` files was added in version 6.25.

gma-spellbook_cost(6)

NAME

gma spellbook_cost – Calculate the value of a wizard's spellbook

SYNOPSIS

```
gma spellbook_cost
```

DESCRIPTION

This little script interactively prompts for the number of spells of each level that appear in a spellbook. It then produces three values: the cost a wizard would need to pay to purchase a blank spellbook and personally scribe the spells into it, the cost you would pay to purchase the already-written spellbook at retail value, and the value you would receive if you found the spellbook and sold it to a dealer.

AUTHOR

Steve Willoughby / steve@madscience.zone.

gma-xpsplit(6)

NAME

gma xpsplit – Divide up experience points and report

SYNOPSIS

```
gma xpsplit [--speed ⟨speed⟩] [--url ⟨base⟩]
```

DESCRIPTION

`Xpsplit` is used to add up the experience points (XP) earned by the party during a game session, so the GM can determine an appropriate XP award to give the party.

The design of `xpsplit` is based on the approach of awarding equal XP to all players, based on campaign milestones reached, but informed by the XP actually earned through the party's actions. The final decision as to the XP award is still at the GM's discretion.

The user is interactively prompted to enter the following information:

`Level:`
: The level the party will be as a result of this award.

`Previous XP:`
: The XP total the characters had at the start of the session.

`XP` ⟨*n*⟩ `+`
: This prompt is repeated indefinitely. Each time, enter the XP awarded from a different encounter or creature. The answers to these will simply be added together. The subtotal accumulated so far is indicated as the ⟨*n*⟩ value at each prompt. Enter 0 to end the entry of XP and move on.

The program will then print four suggestions for XP awards to the party, based on dividing the total XP evenly among 4, 5, 6, and 7 PCs. In each case, it will show how far into the level that XP award would put the party, allowing a judgement call to be made as to whether that is appropriate for the milestone the players have reached at this point.

The program will then prompt for the following:

`XP award:`
: Enter the number of XP to award to the PCs. The program will indicate the percentage through the current level this will place the PCs, as well as the tier that corresponds to. The tier is for staggered advancement, which provides incremental advancement effects at 25%, 50%, and 75% of the way through a level. It will ask if this is ok. If

you answer "y" the XP award is accepted; otherwise this prompt will be repeated for you to try another XP award value until you find one you like.

`Session number:`
: The number of this session in your campaign.

`Session title:`
: What you want to title this session.

`Session date as yyyymmdd:`
: The session date in the format requested.

`Session date as dd-mm-yyyy:`
: The session date in the format requested.

`link to description page:`
: URL to a webpage you use to record your session summaries online.

`Previous tier (0 if was not this level last time):`
: The staggered advancement tier the party was at the last time XP was awarded, as a percentage (0, 25, 50, or 75).

The output of the program will be an HTML-formatted description of the XP award, suitable to post on your game's website or forum. This shows the XP awarded, instructions for staggered advancement options available, and links to the game summary page.

An image called ⟨*base*⟩`/xp`⟨*pct*⟩`.png` is referenced which should show a visual indication of the percentage into the current level they party is now.

Forcing Progress Percentage

As an alternative to adding up XP awards and then choosing a number of XP to award based on that, you can simply tell `xpsplit` to award XP that would take the party to a certain percentage of the way into a level. This is useful if, say, you want the party to be 50% of the way into level 9 because of a story milestone reached, without needlessly figuring out on your own how many XP that would take to accomplish.

To do this, enter the desired level and previous XP as described above, and then enter 0 for the earned XP.

When prompted for the XP award, enter the percentage instead of the number of XP (e.g., enter "`50%`"). Then answer the other prompts as described above.

OPTIONS

`--speed=`⟨*speed*⟩
: Set the advancement speed. Values for ⟨*speed*⟩ may be `slow`, `medium`, or `fast`. The default is `medium`.

`--url=`⟨*base*⟩
: Uses ⟨*base*⟩ as the initial part of the URL included in the repot for the progress image indicator. Defaults to `/gpbp/images`.

AUTHOR

Steve Willoughby / steve@madscience.zone.

BUGS

This is a program written for convenience for the author's personal campaign and is included here in the hopes it may be useful to other GMs but some customization may be desired for other groups.

gma(6)

NAME

gma – Game Master's Assistant commands and main GM console

SYNOPSIS

```
gma --help
gma [--network ⟨name⟩] [--preferences ⟨path⟩] [--world ⟨name⟩]
gma ⟨gma-command⟩ [⟨args⟩...]
gma go ⟨go-gma-command⟩ [⟨args⟩...]
gma man [go] [⟨section⟩] ⟨entry⟩
```

DESCRIPTION

All of the programs and scripts that together comprise GMA may be accessed via one overarching command simply called gma.

This consolidates all the logic needed to find where the various libraries and script interpreters might be on your system in one place and simplifies the operation of GMA. If you wish, you could even make only the single gma script visible in your command search path, using it to locate all the subordinate GMA programs. (You don't normally need to do that, but you *could* if, for example, the names of GMA utilities conflict with the names of other programs you also have in your path.)

The gma script will try to locate all the necessary things it needs if possible, but if it can't, or you want it to use a particular script interpreter for Perl, Python, or Tcl, you can control gma's behavior by setting environment variables (see the ENVIRONMENT section below).

Invoking the Main GMA GM Console

The main tool offered by the GMA Core tool suite is the GM Console, where the GM keeps track of time, events, health stats of everyone, and so forth. This is invoked simply by running gma with no other options:

```
gma
```

Full documentation on this tool appears below in the GM CONSOLE section. But first, let's see how to use the overall gma command to invoke all the other tools.

Invoking Core GMA Utilities

GMA Core comes with a number of utilities located in the bin directory, such as

```
charsheet
```

```
crxp
export-dice-presets
forumpost
initiative-clock
lockfile*
map-id
map-lock
map-reduce
map-translate
mapper
mapserver-etl
markup
mkmap
namegen
papertokens
preferences*
printchar
printmon
rendersizes
roll.py
rotate
servercheck
session-publish
show
skillranks
spellbook_cost
xpsplit
```

Each of these may be invoked by putting the command name as the first parameter of `gma`. If there is a suffix (extension) on the file name as it appears in the `bin` directory, omit it when using `gma` to invoke that utility. For example, even though there is a file called `bin/roll.py`, you would still invoke it by typing `gma roll`.

Programs listed with an asterisk (*) do not appear in the `bin` directory, but are available as sub-commands of the `gma` command.

Any command-line options or parameters to that specific utility may be listed after that. For example:

```
gma printchar --compact-text mychar.gcr
```

In general, given any subcommand `cmd`, the program `$GMA/bin/`⟨*cmd*⟩ will be executed, using whatever script interpreter, environment variables, and so forth, are necessary for that particular command.

More detailed usage instructions for each of those commands may be found in their respective manual entries (e.g., `gma-printchar`(6)).

Invoking the Mapper Client

The gma-mapper(6) program is available separately from GMA Core, since it is often used by players who don't need the rest of GMA. Assuming you have installed `mapper` where `gma` can find it (or used the `$GMA_MAPPER` environment variable to tell `gma` where it is), you can just invoke

`gma mapper` [⟨*args...*⟩]

like the other utility commands and `gma` will launch it for you.

Invoking Go-GMA Utilities

A number of utilities, including the current implementation of the GMA mapper server, have been rewritten in the Go language and distributed as the package `go-gma` separately from the GMA Core.

As of this writing, that includes the following:

`coredb image-audit map-console map-update markup preset-update roll server session-stats upload-presets`

You may use `gma` to invoke them by putting the word `go` before the utility name and arguments. For example:

```
gma go map-console -host example.com -port 12345
```

Printing GMA Manual Entries

If you set up your system to incorporate the GMA manual page files into your man(1) system, just running the usual

`man` [⟨*section*⟩] ⟨*command*⟩

command will work the same as it does for any operating system command. However, the GMA manual pages are not installed system-wide by default. The `gma` program knows how to find the GMA-specific manual page documents, however, so running

`gma man` [⟨*section*⟩] ⟨*command*⟩

will print manual entries for GMA Core, GMA-Mapper, and Go-GMA programs. It will automatically try adding "`gma-`" to the ⟨*command*⟩ name, so the command `gma man foo` will work to display the manual entry for the GMA subcommand `foo` whether the actual manpage is called `foo` or `gma-foo`.

If it is unable to find the GMA manual entry on its own, `gma` will defer to the system man(1) program to see if it can handle it. Therefore, `gma man` will also print arbitrary system manual pages.

In the case where a command in GMA Core and Go-GMA have the same name, the one in GMA Core will be printed. If you want to get the go version, add the keyword `go`:

`gma man go` [⟨*section*⟩] ⟨*command*⟩

ENVIRONMENT

You may set the following environment variables to assist `gma` in finding the necessary resources on your system, or to explicitly force it to choose a par-

ticular one where there may be a choice of, e.g., several different versions of the same language interpreter.

`$GMA`
: This specifies the top-level directory of the GMA Core. Typically this is the place where you cloned the `gma` repository from git. By default, `gma` will assume `$GMA` to be the directory above the one where the `gma` script itself is running from.

`$GMA_IGNORE_GO_GMA`
: If this is set to a non-empty value, it tells `gma` not to consider it an error if it can't find the Go-GMA directory tree installed on the system. You should set this if you installed GMA Core but not Go-GMA.

`$GMA_IGNORE_TCLSH`
: If this is set to a non-empty value, it tells `gma` not to consider it an error if it can't find the `tclsh`(1) interpreter program. You should set this if your system does not have `tclsh` installed and you don't need to run any GMA utility which depends upon it.

`$GMA_IGNORE_WISH`
: If this is set to a non-empty value, it tells `gma` not to consider it an error if it can't find the `wish`(1) interpreter program. You should set this if your system does not have `wish` installed and you don't need to run any GMA utility which depends upon it.

`$GMA_GO_GMA`
: This specifies the top-level directory of Go-GMA. Typically this is the place where you cloned the `go-gma` repository from git. By default, `gma` will assume this is a sibling of the GMA Core directory (i.e., `$GMA/../go-gma`).

`$GMA_MAPPER`
: This specifies the top-level directory of GMA-Mapper. Typically this is either the directory where you cloned the `gma-mapper` repository from git, or the directory where you unpacked the mapper program from zip or tar archive. The `bin` directory where the mapper's program files reside should be accessible via the path `$GMA_MAPPER/bin`.

`$GMA_PERL`
: The name of the Perl interpreter to use when running some GMA utilities. For best results this should be the full pathname to the correct version of the Perl interpreter; otherwise the first command with that name found in your `$PATH` will be executed. By default, `gma` will use the first program called `perl` in your path.

`$GMA_PYTHON`
: The name of the Python interpreter to use when running the GMA utilities. For best results this should be the full pathname to the correct version of the Python interpreter; otherwise the first command

with that name found in your `$PATH` will be executed. By default, `gma` will use the first command called `python3` in your path.

`$GMA_TCLSH`
: The name of the Tcl interpreter to use when running some non-GUI GMA utilities. For best results this should be the full pathname to the correct version of the `tclsh` (Tcl Shell) program; otherwise the first command with that name found in your `$PATH` will be executed. By default, `gma` will use the first program called `tclsh8.6` in your path.

`$GMA_WISH`
: The name of the Tcl/Tk interpreter to use when running some GUI GMA utilities (notably the mapper). For best results this should be the full pathname to the correct version of the `wish` (Windowing Shell) program; otherwise the first command with that name found in your `$PATH` will be executed. By default, `gma` will use the first program called `wish8.6` in your path.

GM CONSOLE

The main `gma` program, which you start by typing the command `gma` without a sub-command, is the main GM console to keep track of various important facts during a game session. It primarily focuses on timekeeping duties, which includes timing of events, durations of effects, and initiative order during combat.

Options

The following options may be given on the command-line to alter the operation of the `gma` program:

`-C, --config` ⟨*filename*⟩
: THIS IS NOW DEPRECATED and is no longer available for use as of GMA version 6.15. It was used by previous versions of GMA to load the legacy configuration file. Since 6.15, GMA now uses the new preferences JSON file (see the `--preferences` option below).

`-h, --help`
: Print a summary of how `gma` may be started, and exit. Note that this will also print the usage information for the overall `gma` command, including how to use sub-commands, as well as the GM console itself.

`-n, --network` ⟨*name*⟩
: Use the named network profile from the preferences file without making that selection permanent (unless you go on to save the preferences to disk from the running program). By default, the network profile that was selected in the preferences file is used.

`-p, --preferences` ⟨*path*⟩
: Instead of the default `~/.gma/preferences.json` file, use the file at the designated ⟨*path*⟩ as the preferences file for GMA.

`-w, --world` ⟨*name*⟩
: Use the named campaign settings from the preferences file without making that selection permanent (unless you go on to save the preferences to disk from the running program). By default, the campaign world name that was selected in the preferences file is used.

Setup

The first time you run GMA on a new system, it won't have the preferences file that it needs. Start by running `gma preferences` to set up your preferences the first time, then run `gma`.

Operation

The operation of the GMA program is far too involved to effectively describe in a brief document such as a manpage. Please refer to the ⟨*GMA Game Master's Guide*⟩ manual for all the details, and step-by-step tutorials.

SEE ALSO

`gma-charsheet`(6), `gma-go-coredb`(6), `gma-crxp`(6), `gma-export-dice-presets`(6), `gma-initiative-clock`(6), `gma-lockfile`(1), `gma-go-map-console`(6), `gma-map-id`(6), `gma-map-lock`(6), `gma-map-reduce`(6), `gma-map-translate`(6), `gma-go-map-update`(6), `gma-mapper`(5), `gma-mapper`(6), `gma-mapserver-etl`(6), `gma-mkmap`(6), `gma-namegen`(6), `gma-preferences`(6), `gma-go-preset-update`(6), `gma-printchar`(6), `gma-printmon`(6), `gma-rendersizes`(6), `gma-rotate`(6), `gma-roll`(6), `gma-go-roll`(6), `gma-go-server`(6), `gma-servercheck`(6), `gma-go-session-stats`(6), `gma-show`(6), `gma-skillranks`(6), `gma-spellbook_cost`(6), `gma-go-upload-presets`(6), and `gma-xpsplit`(6).

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

The `gma` program began life decades ago, in the 1990s, as a terminal-based C++ program and has evolved significantly since then. This document describes the version 6.x implementation.

BUGS

APPENDIX E

Go-GMA Game Program Reference

> Gentlemen should not waste
> their time on trivial
> games—they should study go.
>
> —Confucius, The Analects, ca.
> 500 B.C.E.

The following pages document the command-line (CLI) invocation and usage of the programs that come with the Go-GMA package. Go-GMA is a rewrite of some of the core GMA utilities and much of the API from Python to the Go language. Most significantly, this provides the most stable and featured version of the GMA Mapper Server to date. The older Python server which is part of GMA Core is now deprecated in favor of the Go version.

As with the previous section, the documentation for these commands is placed here in section 6 because these are game-playing programs.

These manual entries are also available on the system where GMAis installed in the form of online manual pages ("manpages"), which can be viewed using the `man` command of GMA (e.g., "`gma man go server`"). These pages are in section 6, so following the conventions for Unix manpages, their names have "(6)" appended to them.

gma-go-coredb(6)

NAME

gma go coredb – import/export entries in the GMA core SRD database

SYNOPSIS

(If using the full GMA core tool suite)

`gma go coredb` [options as described below...]

(Otherwise)

`coredb -h`

`coredb -help`

`coredb` [`-D` ⟨*flags*⟩] [`-e` ⟨*file*⟩] [`-f` [`!`]⟨*regexp]*⟩ [`-I`] [`-i` ⟨*file*⟩] [`-l` ⟨*file*⟩] [`-preferences` ⟨*file*⟩] [`-srd`] [`-t` ⟨*list*⟩]

`coredb` [`-debug` ⟨*flags*⟩] [`-export` ⟨*file*⟩] [`-filter` [`!`]⟨*regexp]*⟩ [`-ignore-case`] [`-import` ⟨*file*⟩] [`-log` ⟨*file*⟩] [`-preferences` ⟨*file*⟩] [`-srd`] [`-type` ⟨*list*⟩]

DESCRIPTION

`Coredb` is used for maintenance of the core (SRD) database. Once the initial load of community-published SRD data is performed via `gma initdb`, you will probably add a number of your own entries to the database for your own campaign. These are known as "local" entries, while the ones that are in the core game's SRD content which are licensed to be freely used (according to the applicable license) are known as "SRD" entries.

If run with the `-export` option, `coredb` will export the local entries from your core database to the specified output file, encoded in JSON. If run with the `-import` option, the opposite is done: the JSON-encoded file is read, and its contents added to the core database. If this file's contents match the name or ID (as appropriate) of an existing entry, that entry is updated in-place in the database.

OPTIONS

The command-line options described below have a long form (e.g., `-log`) and a short form (e.g., `-l`) which are equivalent. In either case, the option may be introduced with either one or two hyphens (e.g., `-log` or `--log`). Options

which take parameter values may have the value separated from the option name by a space or an equals sign (e.g., `-log=`⟨*path*⟩ or `-log` ⟨*path*⟩), except for boolean flags which may be given alone (e.g., `-I`) to indicate that the option is set to "true" or may be given an explicit value which must be attached to the option with an equals sign (e.g., `-I=true` or `-I=false`).

You *may not* combine multiple single-letter options into a single composite argument, (e.g., the options `-I` and `-h` would need to be entered as two separate options, not as `-Ih`).

`-D`, `-debug` ⟨*flags*⟩
: This adds debugging messages to `coredb`'s output. The ⟨*flags*⟩ value is a comma-separated list of debug flag names, which may be any of the following:

`all`	Enable all possible debugging flags.
`none`	If you want to explicitly disable debugging, you can specify `none` for the ⟨*flags*⟩ value to override the previous flag list and effectively disable debugging. If `none` appears in a comma-separated flag list, it cancels all the previously-set flags, but any other flag names which occur after it will be set.
`misc`	Show miscellaneous debugging messages.
`queries`	Show SQL queries made to the database.

`-e`, `-export` ⟨*file*⟩
: Export the local entries from the database to the named JSON-encoded ⟨*file*⟩.

`-f`, `-filter` `[!]`⟨*regexp*⟩
: When importing or exporting, only include entries matching the regular expression ⟨*regexp*⟩. If ⟨*regexp*⟩ begins with a "`!`" character, only entries which do *not* match the expression are included.

  The ⟨*regexp*⟩ is matched against the `Code` and `Name` fields of the entries in question. If either matches, then the entry is included (or excluded). For language entries, the `Language` field is checked instead. For monsters in the bestiary, the `Code` and `Species` fields are checked.

`-I`, `-ignore-case`
: Pattern matching via the `-filter` option should be done irrespective of case.

`-i`, `-import` ⟨*file*⟩
: Read the contents of the JSON-encoded ⟨*file*⟩, importing them to the database.

`-l`, `-log` ⟨*file*⟩
: Directs log messages (including any debugging output) to the named

⟨*file*⟩ instead of the standard output. You may explicitly specify the standard output by using a single hyphen as the file name (e.g., "`-log=-`").

`-preferences` ⟨*file*⟩
: Normally, `coredb` reads the default GMA preferences file to obtain the pathname of the core database file. This option specifies an alternative preferences file from which to get that pathname.

`-srd`
: Instead of exporting only the local entries and assuming imported entries are local, export SRD entries and assume imported entries are SRD.

`-t`, `-type` ⟨*list*⟩
: Entries exported will include only database entries of the specified type(s). When importing, any records in the import file which are not of the specified type(s) are skipped over. The ⟨*list*⟩ value is a comma-separated list of type names, which may be any of the following:

`all`	All entry types (the default).
`none`	No types. If `none` appears in a comma-separated type list, it cancels all the previously-set types, but any other type names which occur after it will be set.
`bestiary`	Monster entries. (The names `creature`[`s`] and `monster`[`s`] are also recognized.)
`class`[`es`]	Character class entries.
`feat`[`s`]	Feats.
`language`[`s`]	Languages.
`skill`[`s`]	Skills.
`spell`[`s`]	Spells.
`weapon`[`s`]	Weapons.

FILE FORMAT

The JSON file used for exporting and importing operations is as documented for the `util.CoreImport` function of the Go-GMA project (see pkg.go.dev/github.com/MadScienceZone
gma/v5/util#CoreImport Each of the data structures marshalled into JSON for monsters, weapons, etc., are documented on the same page at that URL.

SEE ALSO

`gma`(6), `gma-importdb`(6), `gma-initdb`(6).

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

The core database appeared in GMA Core version 7.0; this utility appeared in go-gma version 5.7.0.

BUGS

gma-go-image-audit(6)

NAME

gma go image-audit – GMA server image accuracy audit

SYNOPSIS

[gma go] `image-audit` [`-delete`] [`-help`] [`-list`] `-sqlite` ⟨*path*⟩ `-webroot` ⟨*path*⟩

DESCRIPTION

`Image-audit` is run on a system which hosts both a GMA server and the web server which offers the map image tiles to mapper clients (or at least has filesystem access to those files).

It checks for any files in the server's database (which means they will be offered by the server to any clients asking about those images) which do not have actual image files being served for them.

It is important that the server doesn't advertise images to any mapper clients which don't actually exist on the server. This will cause the mappers to report errors when the download of the images fail, and eventually the server may kick them off when their repeated requests for the same image files is interpreted as a DoS attack.

N.B. The `image-audit` program can't check for, or correct, everything that could go wrong. For example, it will be satisfied if any image file exists to satisfy requests for the corresponding database entry. However, it does not check to ensure that this image file is available in all (or any) specific graphic format (e.g., GIF or PNG) that your map clients will request. You must still make sure the files are provided in all formats your clients care about.

Also, in the case of animated images, no check is made that all expected frames are available or that any individual frame files exist; just that there is some image file at the advertised location.

OPTIONS

The command-line options described below may be introduced with either one or two hyphens (e.g., `-delete` or `--delete`). Options which take parameter values may have the value separated from the option name by a space or an equals sign (e.g., `-sqlite=mygame.db` or `-sqlite mygame.db`), except for boolean flags which may be given alone (e.g., `-delete`) to indicate that the option is set to "true" or may be given an explicit value which must be attached to the option with an equals sign (e.g., `-delete=true` or `-delete=false`).

`-delete`
: When images are defined in the server's database but no corresponding image file is actually found, actually delete the image's definition from the server's database file so it won't continue to tell clients that it knows their location. If you update the database later (say, by loading a map file which defines images), be sure the web server actually has those files available for download at that point.

`-list`
: In addition to messages normally printed about missing files or removed database entries, this option causes `image-audit` to name all the files that don't seem to be in the right place or have names not conforming to the expected format (as opposed to just reporting how many of them were found), and to list all known files (whether mentioned in the database or discovered in the web server's GMA image directory tree), along with their status. The status is indicated by a set of single letter flags:

  `?` The file's type could not be determined.

  `I` This is an image file.

  `M` This is a map file.

  `c` This file is known to the server but stored by a client, not on the web server.

  `d` This file is known in the database and should be stored on the web server.

  `w` This file was found on the web server.

`-h`, `-help`
: Print a usage summary and exit.

`-sqlite` ⟨*path*⟩
: Read the file specified by ⟨*path*⟩ as the a sqlite database the server uses for its persistent storage (as specified to the server program's own `-sqlite` option). `Image-audit` may be run while the server is also accessing this file, but note that updating the database by either program may briefly lock the other out from accessing it. Therefore, it is best to run `image-audit` when the server is shut down or at least quiescent.

`-webroot` ⟨*dir*⟩
: Consider all the files from dir down to be the directory structure behind what the GMA mapper client knows as the "image base URL". Thus, a server image ID of "`abcdef`" corresponds to disk file ⟨*dir*⟩`/a/ab/abcdef.png`, et al.

SEE ALSO

`gma`(6), `gma-mapper`(5), `gma-go-server`(6). `gma-mapper`(6),

AUTHOR

Steve Willoughby / steve@madscience.zone.

BUGS

There are more checks that `image-audit` could make to do more to evaluate the integrity of the server's image storage, but this is a first start.

gma-go-map-console(6)

NAME

gma go map-console – GMA map service console (Go version)

SYNOPSIS

(If using the full GMA core tool suite)

`gma go map-console` [options as described below...]

(Otherwise)

`map-console -h`

`map-console -help`

`map-console` [`-Dm`] [`-C` ⟨*configfile*⟩] [`-c` ⟨*calendar*⟩] [`-H` ⟨*host*⟩] [`-l` ⟨*logfile*⟩] [`-list-profiles`] [`-P` ⟨*password*⟩] [`-p` ⟨*port*⟩] [`-S` ⟨*profile*⟩] [`-u` ⟨*user*⟩]

`map-console` [`-calendar` ⟨*calendar*⟩] [`-config` ⟨*configfile*⟩] [`-debug`] [`-help`] [`-host` ⟨*host*⟩] [`-list-profiles`] [`-log` ⟨*logfile*⟩] [`-mono`] [`-password` ⟨*password*⟩] [`-port` ⟨*port*⟩] [`-select` ⟨*profile*⟩] [`-username` ⟨*user*⟩]

DESCRIPTION

`Map-console` provides a way to interact directly with the GMA game server. It will print any server messages it receives in a colorized text representation. Commands typed into `map-console` are sent to the server as described in detail below.

This tool is designed primarily for debugging the server. Its input and output is not designed to be user-friendly, but rather to make it possible for someone familiar with the server's operation and network protocol to manually manipulate it.

When `map-console` starts, it will look for a GMA Mapper preferences file at `~/.gma/mapper/preferences.json`. If present, that file will be loaded to obtain the connection information. If no such file was found, it will fall back to the legacy behavior by reading an old-style Mapper configuration file at `~/.gma/mapper/mapper.conf`.

If an old-style configuration file is named explicitly via the `-config` CLI option, it will be loaded after `preferences.json` (such that its settings override the preferences file's information).

OPTIONS

The command-line options described below have a long form (e.g., `-port`) and a short form (e.g., `-p`) which are equivalent. In either case, the option may be introduced with either one or two hyphens (e.g., `-port` or `--port`). Options which take parameter values may have the value separated from the option name by a space or an equals sign (e.g., `-port=2323` or `-port 2323`), except for boolean flags which may be given alone (e.g., `-m`) to indicate that the option is set to "true" or may be given an explicit value which must be attached to the option with an equals sign (e.g., `-m=true` or `-m=false`).

You **may not** combine multiple single-letter options into a single composite argument, (e.g., the options `-r` and `-m` would need to be entered as two separate options, not as `-rm`).

Any of these options will override the settings read from `preferences.json` or `mapper.conf`.

`-c`, `-calendar` ⟨*name*⟩
: This specifies the name of the calendar system in use for the campaign. If not specified, it defaults to "`golarion`". Normally, the server should be configured to tell all clients (including `map-client`) what calendar is in use, in which case if you also provide this option it will override the server's advertised calendar in favor of the one you are explicitly setting here.

`-C`, `-config` ⟨*file*⟩
: The named ⟨*file*⟩ is read to set the same options as documented here for command-line parameters. The only difference is that the long name of the option must be used with no leading hyphens, and with equals signs between the option name and value, one option per line. For example, the file could contain a line reading

```
host=example.com
```

but not lines like

```
-host=example.com
--host example.com
h=example.com
```

Any options set on the command line override those read from the configuration file. To set a boolean flag to be true, simply name it on a line of the file. For example:

```
mono
```

Lines beginning with an octothorpe ("`#`") are ignored as comments.

`-D`, `-debug` ⟨*flags*⟩
: This adds debugging messages to `map-console`'s output. The ⟨*flags*⟩ value is a comma-separated list of debug flag names, which may be any of the following:

`all`	Enable all possible debugging flags.
`none`	If you want to explicitly disable debugging (e.g., if debugging is enabled in your configuration file but you want to turn it off via command-line option), you can specify `none` for the ⟨*flags*⟩ value to override the previous flag list and effectively disable debugging. If `none` appears in a comma-separated flag list, it cancels all the previously-set flags, but any other flag names which occur after it will be set.
`auth`	Authentication operations
`binary`	When printing certain values such as data transmitted over the network, include a hex dump of the actual binary data in addition to other debugging messages.
`events`	Show background events such as expiring timers and received process signals.
`i/o`	Show input/output operations used to get data in and out of the client.
`messages`	Show the server messages sent and received.
`misc`	Show miscellaneous debugging messages

`-H`, `-host` ⟨*host*⟩
: Specifies the server's hostname.

`-l`, `-log` ⟨*file*⟩
: Directs log messages (including any debugging output) to the named ⟨*file*⟩ instead of the standard output. You may explicitly specify the standard output by using a single hyphen as the file name (e.g., "`-log=-`").

`-list-profiles`
: Prints a list of all the profiles defined in the mapper preferences and exits.

`-m`, `-mono`
: Prevent `map-console` from using ANSI escape codes to colorize the output. With this flag set, only plain text output will be emitted.

`-P`, `-password` ⟨*password*⟩
: If the server has authentication enabled, this specifies the password to be sent to log in. If the GM password is given, then the `map-console` user will have GM privileges on the server; otherwise normal player privileges will be granted, as with any client.

`-p`, `-port` ⟨*port*⟩
: Specifies the server's TCP port number. The GMA map server's default port, 2323, will be assumed by default.

`-S`, `-select` ⟨*profile*⟩
: Use the named ⟨*profile*⟩ from the mapper preferences for the connection information instead of the one currently designated as the mapper's current profile.

`-u`, `-username` ⟨*user*⟩
: This specifies the user name by which the server will know you. If you log in with the `GM` credentials, the server will assign you the name "`GM`" regardless of what you request here. If you don't use the `GM` credentials, you may not ask for the name "`GM`". If you do not specify a username, it will default to your local system username if possible.

COMMANDS

Commands typed into the standard input of `map-console` are sent to the server as described here.

Obviously, this should be done with caution by someone intimately familiar with the protocol and who understands the implications of injecting commands into the working system like this.

Pre-Defined Commands

The following commands are recognized with a simple interface which should be easier to type than a full JSON string would be. They may not suffice for every possible set of operations; they are designed to handle common cases conveniently.

Note that the command names may be typed in either upper- or lowercase, but the values are taken exactly as typed. The entire input line must conform to the syntax of a Tcl list string. This means, in a nutshell, that the command and its arguments are separated by spaces, and that multiword values need to be enclosed in curly braces so they are interpreted as a single value. Braces must be balanced. An empty string value may be typed as "`{}`".

`AI` ⟨*name*⟩ ⟨*size*⟩ ⟨*file*⟩
: Upload an image from a local named file, for clients to access with the given ⟨*name*⟩ and zoom factor ⟨*size*⟩ (the latter expressed as a real number with 1.0 meaning the normal zoom setting).

  This is deprecated. Instead, images should be prepared using the `gma rendersizes`(6) program and uploaded to the server directly.

`AI?` ⟨*name*⟩ ⟨*size*⟩
: Ask the server and/or other connected clients if they know where an image file with the given ⟨*name*⟩ and zoom factor ⟨*size*⟩ may be found.

`AI@` ⟨*name*⟩ ⟨*size*⟩ ⟨*id*⟩
: Inform the server that it should advertise the location of a stored image file with the given ⟨*name*⟩ and ⟨*size*⟩ as the server storage name

⟨*id*⟩. (Refer to the full documentation for an explanation of what that actually means.)

`AV` ⟨*label*⟩ ⟨*x*⟩ ⟨*y*⟩
: Adjust the view of all clients so that grid label ⟨*label*⟩, or if that is empty or unable to be understood, scroll so that the display is the fraction ⟨*x*⟩ of the way to the right and ⟨*y*⟩ of the way down, where ⟨*x*⟩ and ⟨*y*⟩ are numbers ranging from 0.0 (far left or top) to 1.0 (far right or bottom).

`CC` ⟨*silent?*⟩ ⟨*target*⟩
: Tell the server to clear the chat history. If ⟨*silent?*⟩ is true, do so without announcement. The ⟨*target*⟩ is negative, all messages are deleted except for the most recent –⟨*target*⟩ messages. Otherwise, all messages with IDs less that ⟨*target*⟩ are deleted.

`CLR` ⟨*id*⟩
: Remove the specified object from the map clients. The ⟨*id*⟩ may be an object ID number, name as known to the mapper (e.g., "`Bob`" or "`goblinMimg=Goblin`"), or the values `*`, `E*`, `M*`, or `P*`, to remove all objects, all map elements, all monsters, or all player tokens, respectively.

`CLR@` ⟨*id*⟩
: Clears all elements that are mentioned in the server-side map file with the specified ⟨*id*⟩.

`CO` ⟨*enabled?*⟩
: Enter combat mode if ⟨*enabled?*⟩ is true, otherwise exit to normal play mode.

`D` ⟨*recipients*⟩ ⟨*rollspec*⟩ [⟨*id*⟩]
: Ask the server to roll the dice as specified in ⟨*rollspec*⟩ with the results being sent to the list of names in ⟨*recipients*⟩ (which is itself a brace-enclosed, space separated Tcl list). The special recipient names * and % may be used to send the results to all clients, or blindly send them only to the GM, respectively. In this case the * or % must be the only thing in the recipient list. You may optionally provide an arbitrary ⟨*id*⟩ which will be sent back with the die-roll results from the server.

`DD` ⟨*list*⟩
: Set your server-side die-roll preset list to ⟨*list*⟩, which is a brace-enclosed list of presets, each of which is a brace-enclosed list of three values: preset name, description, and the die-roll spec for that roll.

`DD+` ⟨*list*⟩
: Just like `DD` but adds the contents of ⟨*list*⟩ to your existing set of presets rather than replacing them.

`DD/` ⟨*re*⟩
: Delete all the die-roll presets stored for you whose names match the regular expression ⟨*re*⟩.

`DR`
: Request that the server send you all your die-roll presets.

`EXIT`
: Exit the `map-console` program.

`HELP`
: Prints out a command summary.

`L` ⟨*filename*⟩
: Load the map file stored in a local file onto all the connected map clients. This replaces any existing elements on the map previously.

`L@` ⟨*id*⟩
: Load the map file stored on the server under the given ⟨*id*⟩ onto all the connected map clients. This replaces any existing elements on the map previously.

`M` ⟨*filename*⟩
: Like `L` but merges the contents of the map with the existing contents of the map instead of replacing them.

`M?` ⟨*id*⟩
: Tells clients to pre-fetch and cache a copy of the server's map file stored under the given ⟨*id*⟩.

`M@` ⟨*id*⟩
: Like `L@` but merges the map contents with the existing contents of the map.

`MARK` ⟨*x*⟩ ⟨*y*⟩
: Visibly mark the given (⟨*x*⟩,⟨*y*⟩) coordinates on the map for a second.

`OA` ⟨*id*⟩ ⟨*kvlist*⟩
: Set one or more attributes of the object with the given ⟨*id*⟩ to those in ⟨*kvlist*⟩. The latter is a brace-enclosed, space-separated Tcl list where the first value is the name of an attribute, the next is the value for that attribute, and so on for each pair of attribute names and values you need to change.

`OA+` ⟨*id*⟩ ⟨*attribute*⟩ ⟨*list*⟩
: For object attributes whose values are a list of strings, this command adds one or more values to that object's attribute.

`OA-` ⟨*id*⟩ ⟨*attribute*⟩ ⟨*list*⟩
: Like `OA+` but removes each of the values in ⟨*list*⟩ from the named attribute.

`POLO`
: Sends a client response to the server's `MARCO` ping message. `Map-client` automatically sends these every time the server pings it.

`PS` ⟨*id*⟩ ⟨*color*⟩ ⟨*name*⟩ ⟨*area*⟩ ⟨*size*⟩ `player|monster` ⟨*x*⟩ ⟨*y*⟩ ⟨*reach*⟩

: Place a creature token on the map.

`QUIT`
: Synonymous with `EXIT`.

`SYNC`
: Request that the server send a full dump of the game state to you.

`SYNC CHAT [`⟨*target*⟩`]`
: Request that the server send a full dump of all chat messages in its history to you. If ⟨*target*⟩ is given, it limits the number of requested chat message. If it is negative, only the most recent –⟨*target*⟩ messages are sent. Otherwise, only any messages with message IDs greater than ⟨*target*⟩ are sent.

`TO` ⟨*recipients*⟩ ⟨*message*⟩
: Send a chat ⟨*message*⟩ to the users named in the ⟨*recipients*⟩ list. The latter value may be given as described above for the `D` command.

`/CONN`
: Request a list of all connected clients.

`?`
: Synonymous with `HELP`.

Generalized Command Entry

You may also type a full command with options by typing "`!`⟨*command*⟩" followed by a number of parameters in the following forms. The same rules apply as above, so values which contain spaces will need to be enclosed in braces.

⟨*key*⟩=⟨*val*⟩
: Include parameter ⟨*key*⟩ in the JSON payload for the command, with the value ⟨*val*⟩ as a character string.

⟨*key*⟩#⟨*val*⟩
: As above, but don't quote ⟨*val*⟩ as a string. Thus, ⟨*val*⟩ may be a number, `true`, `false`, or `null`.

⟨*key*⟩:⟨*val*⟩
: As ⟨*key*⟩=⟨*val*⟩ except that any underscore characters in ⟨*val*⟩ are converted to spaces, making it unnecessary to put braces around this parameter.

For example, typing the command:

```
!d RequestID=abc123 ToAll#true RollSpec:d20+2_acid
```

will send this command to the server:
`D {"RequestID":"abc123","ToAll":true,"RollSpec":"d20+2 acid"}`
The ⟨*command*⟩ name may be typed in any case but the parameters must be typed exactly as expected by the server protocol.

Raw Message Entry

Finally, it is also possible to simply type a literal string of characters which will be sent to the server AS-IS without further interpretation.

This is done by prefixing the string with a backquote character. Thus, the previous server command could have been typed into `map-console` literally as:

```
‘D {"RequestID":"abc123","ToAll":true,"RollSpec":"d20+2
acid"}
```

SEE ALSO

gma(6), gma-mapper(5), gma-mapper(6).

AUTHOR

Steve Willoughby / steve@madscience.zone.

BUGS

This program blindly assumes that the user's terminal understands ANSI standard escape codes to produce colored text (although the `-mono` option will stop it from doing so.)

gma-go-map-update(6)

NAME

gma go map-update – update the format of a saved GMA map file

SYNOPSIS

(If using the full GMA core tool suite)

`gma go map-update` ⟨*file*⟩ ...

(Otherwise)

`map-update` ⟨*file*⟩ ...

DESCRIPTION

`Map-update` reads each ⟨*file*⟩ named on the command line and rewrites the file in the current map file format. A copy of the original is kept with the same name plus a `.bak` suffix.

The map file format changed significantly between format version 17 and 20. The `map-update` program can read format 17 files, so this provides a way to update existing map files to the newer format.

SEE ALSO

`gma`(6), `gma-mapper`(5), `gma-mapper`(6).

AUTHOR

Steve Willoughby / steve@madscience.zone.

gma-go-markup(6)

NAME

gma go markup – Render GMA markup text into plain text, PostScript, or HTML (Go version)

SYNOPSIS

(If using the full GMA tool suite)
`gma go markup` [options as described below…]
(otherwise)
`markup -help`
`markup` [`-html`] [`-preamble`] [`-ps`] [`-syntax`] < ⟨*input*⟩ > ⟨*output*⟩

DESCRIPTION

GMA uses a simple markup syntax described in `gma-markup`(7) for free-form text content. The `gma go markup` program documented here is a standalone utility to render text in this format into various output forms.

In normal operation, `markup` acts as a filter, converting the data sent to its standard input to its standard output.

Note that there is a version of this utility that appears in both the Python-based GMA Core (invoked as `gma markup`) and in the Go-GMA project (invoked as `gma go markup`). They serve the same function, but each uses the GMA markup implementation in their respective languages.

OPTIONS

The following options control the behavior of this program.

`-help`
: Print a summary of the command-line usage and exit.

`-html`
: Render the output as HTML instead of the default plain text.

`-preamble`
: Print the GMA PostScript preamble before printing anything else.

`-ps`
: Render the output as PostScript instead of the default plain text. This isn't sufficient for use on a PostScript device without some up-front initialization and inclusion of GMA's usual PostScript preamble, but where free-form markup-formatted text would appear in the PostScript document generated by GMA, this is the content that would be used there.

`-syntax`
: Instead of reading standard input, render a description of the markup syntax as expected by this program.

SEE ALSO

`gma`(6), `gma-go-markup`(6), `gma-markup`(7), `gma-preamble`(7).

AUTHOR

Steve Willoughby / steve@madscience.zone.

gma-go-preset-update(6)

NAME

gma go preset-update – update the format of a saved GMA map file

SYNOPSIS

(If using the full GMA core tool suite)

`gma go preset-update` [⟨*file*⟩ ...]

(Otherwise)

`preset-update` [⟨*file*⟩ ...]

DESCRIPTION

`Preset-update` reads each ⟨*file*⟩ named on the command line and rewrites the file in the current die-roll preset file format. A copy of the original is kept with the same name plus a `.bak` suffix.

If no files are named, `preset-update` reads from its standard input and writes to its standard output.

The file format changed significantly between format versions 1 and 2. The `preset-update` program can read format 1 files, so this provides a way to update existing map files to the newer format.

SEE ALSO

`gma-dice`(5), `gma`(6).

AUTHOR

Steve Willoughby / steve@madscience.zone.

gma-go-roll(6)

NAME

gma go roll – GMA die roller CLI application (Go version)

SYNOPSIS

(If using the full GMA core tool suite)

`gma go roll` [options as described below...]

(Otherwise)

`roll -h`

`roll -help`

`roll -syntax`

`roll` [`-dice` ⟨*string*⟩] [`-json`] [`-seed` ⟨*int*⟩]

DESCRIPTION

`Roll` provides a command-line utility that rolls dice using the GMA dice library without requiring a functioning GMA game server to roll the dice for you.

It can be used interactively by users or it can be embedded into scripts or other back-end services.

The full description of what can go into a die roll expression string is documented in `gma-dice-syntax`(7).

OPTIONS

Options may be introduced with either one or two hyphens (e.g., `-json` or `--json`). Options which take parameter values may have the value separated from the option name by a space or an equals sign (e.g., `-seed=2323` or `-seed 2323`), except for boolean flags which may be given alone (e.g., `-json`) to indicate that the option is set to "true" or may be given an explicit value which must be attached to the option with an equals sign (e.g., `-json=true` or `-json=false`).

`-dice` ⟨*string*⟩

Specify the die-roll expression to be rolled, such as "`3d6`".

Multiple die-roll expressions may be given as the value to this option, separated from one another by semicolons ("`;`"). In this case each will

be rolled in the sequence given. This allows a group of die rolls to be performed from a single seed value in a single invocation of the program.

If this is not given, `roll` will interactively prompt for die-roll expressions and roll each as they're typed in. Typing a blank line repeats the previous expression. The program will exit on EOF.

`-help`
: Print a command option summary and exit.

`-json`
: Output the results as a JSON string instead of plain text.

`-seed` ⟨*int*⟩
: Instead of using a random seed value, base the die roll results on the given value. The ⟨*int*⟩ value is a 64-bit integer expressed in decimal digits.

`-syntax`
: Print a summary of the die-roll expression syntax and exit. In interactive mode, this help text may be produced by typing "`help`" as the input line.

OUTPUT FORMATS

Text Output

The plain-text output format (the default) briefly summarizes each result and the die rolls which were made to reach that result. This output uses ANSI escape codes to add colors for emphasis.

In case a die roll specification yields three or more results (e.g., from a spec such as "`3d6+5|repeat 20`"), a line of statistics will be printed, showing the number of results (N), population mean value (μ), standard deviation (σ), median value (Md), mode value(s) (Mo), and the sum of all results in the set (Σ).

JSON Output

If JSON output is requested, a single JSON object will be printed to standard output, with the following fields. Fields which are not relevant or have a zero value may be omitted entirely from the output.

`ResultSet` *(list of objects)*
: The value is a list of JSON objects, each containing one set of results from the set of die-roll specifications. Each value is an object with the following fields:

`Title` *(string)*
: The title (label) string, if any, specified by the user (this is the initial part of the die-roll spec, separated from the rest of the expression by an equals sign).

`Results` *(list of objects)*
: This is a list of JSON objects, each describing one of the results from a single die-roll specification. Each element of the list is an object with the following fields:

  `ResultSuppressed` *(bool)*
  : If true, this indicates that the results are not shown to the user. They may have been sent privately to the GM, for example,

  `InvalidRequest` *(bool)*
  : If true, this indicates that the request could not be understood and thus no results have been generated.

  `Result` *(int)*
  : The total result of the die-roll expression, truncated to an integer (rounded toward zero).

  `Details` *(list of objects)*
  : This list describes the die-roll expression that led to the result, including subtotals and the value rolled for each individual die. Each element of the list is a JSON object with the following fields:

    `Type`
    : The type of value this element represents.

    `Value`
    : The value of the element, expressed as a string.

`Stats` *(object or null)*
: If there are 2 or more results in the result set, this object will provide statistics about the result set's values. It contains the following fields:

  `N` *(int)*
  : The population size (number of results analyzed).

  `Mean` *(float)*
  : The population mean (simple average).

  `Median` *(float)*
  : The median value of the result set.

  `Mode` *(list*`of`*ints)*
  : The mode value (the value which occured most). If there's a tie for the most popular number, all of the most popular ones are listed.

  `StdDev` *(float)*
  : The standard deviation of the values.

`Sum` *(int)*
: The sum of all the values in the result set.

`Seed` *(int)*
: The random-number seed used to generate this set of die rolls.

SEE ALSO

`dice`(3), `gma-roll`(6).
This program is analogous to, but has more features than, the Python-based `gma roll` program from GMA-Core.

AUTHOR

Steve Willoughby / steve@madscience.zone.

BUGS

The default seed chosen by the underlying die-rolling library is always 32 bits long, but that may change in the future, so don't rely on that fact.

gma-go-server(6)

NAME

gma go server – GMA battle grid map server (Go version)

SYNOPSIS

[gma go] `server` [`-cpuprofile` ⟨*path*⟩] [`-debug` ⟨*flags*⟩] [`-endpoint` [⟨*hostname*⟩]:⟨*port*⟩] [`-help`] [`-init-file` ⟨*path*⟩] [`-log-file` ⟨*path*⟩] [`-password-file` ⟨*path*⟩] `-sqlite` ⟨*path*⟩ [`-telemetry-log` ⟨*path*⟩] [`-telemetry-name` ⟨*string*⟩]

DESCRIPTION

The individual `gma-mapper`(6) clients used by players in a game may keep in contact with one another so that they all display the same contents. A change made on one client (moving a creature token or adding a room, say) appears on all the others. This is accomplished by starting a `server` process and having all of the `mapper` clients connect to it via their `--host` and `--port` options.

Once connected, the server will send an initial greeting that may define a list of player character tokens to appear on the `mapper` context menus, or any other useful information the clients need to have at startup time. It may, at the GM's option, even initialize the client to show the full current game state.

From that point forward, the server relays traffic between the clients, so they communicate with each other via the service. The server also tracks the commands it sees, so that it maintains a notion of the current state of the game. Clients may re-sync with the server in case they restart or otherwise miss any updates so they match the server's state. The server may respond directly to some client queries if it knows the answer rather than referring the query to the other clients.

To guard against nuisance or malicious port scans and other superfluous connections, the server will automatically drop any clients which don't authenticate within a short time. (In actual production use, we have observed some automated agents which connected and then sat idle for hours, if we didn't terminate their connections. This prevents that.)

OPTIONS

The command-line options described below may be introduced with either one or two hyphens (e.g., `-endpoint` or `--endpoint`). Options which take parameter values may have the value separated from the option name by a space or an equals sign (e.g., `-endpoint=:2323` or `-endpoint :2323`).

`-cpuprofile` ⟨*path*⟩
: Enable CPU profiling via Go's pprof tool. Sample data will be saved to the named ⟨*path*⟩.

`-debug` ⟨*flags*⟩
: Add debugging information to the server log. The ⟨*flags*⟩ value is a comma-separated list of debugging options. The following options are available:

`all`	All possible debugging information.
`none`	No debugging information (this cancels any previously-specified debug flags, but more may be added after this).
`auth`	Authentication operations.
`db`	Database operations.
`events`	Background events.
`i/o`	Input/output operations.
`init`	Client initialization.
`messages`	Message traffic between the server and clients.
`misc`	Miscellaneous debugging.
`qos`	Quality of service stats (even if no action was taken).
`state`	Changes to the game state.

`-endpoint` [⟨*hostname*⟩]:⟨*port*⟩
: Accept incoming client connections on the specified ⟨*hostname*⟩ and TCP ⟨*port*⟩ number. If ⟨*hostname*⟩ is omitted, connections are accepted on all the machine's network interfaces. If the entire option is omitted, it defaults to port 2323.

`-h`, `-help`
: Print a usage summary and exit.

`-init-file` ⟨*path*⟩
: The contents of this file are used to initialize the clients every time they connect. See the **CLIENT INITIALIZATION** section below for details.

`-log-file` ⟨*path*⟩
: Append a record of server actions to the specified file. If debugging is enabled, those messages will go to the log file as well. By default, the log is printed to the standard output, which may also be explicitly specified by a single hyphen ("-") as ⟨*path*⟩.

`-password-file` ⟨*path*⟩
: This enables client authentication. By default, the server will allow any client to connect and immediately interact with it. However, if this option is given, the server will require a valid user credential before allowing the client to operate. The contents of the password file are stored in plaintext, one password per line.

  The first line is the general player password. Any client connecting with this credential will be admitted with any username they request other than "`GM`".

  The next line, if present, gives the privileged `GM` password. Any client connecting with this credential will be granted with game master privileges under the username "`GM`".

  Any subsequent lines have the format "⟨*username*⟩:⟨*password*⟩". This assigns a specific ⟨*password*⟩ for the given ⟨*username*⟩, such that any client wishing to sign on with that specific username must present this specific credential.

  N.B. This is an extremely trivial challenge-response authentication mechanism used solely to protect a game server from revealing in-game spoilers and rejecting nuisance connections. It should **not** be relied upon to secure any sensitive information. No passwords used here should be the same as passwords used for anything else of consequence.

`-sqlite` ⟨*path*⟩
: Specifies the filename of a sqlite database the server will use to maintain persistent state. This includes such things as stored die-roll presets, known image locations, and the chat history. If ⟨*path*⟩ does not exist, a new empty database will automatically be created by the server.

`-telemetry-log` ⟨*path*⟩
: If the server is configured to send telemetry metrics, this provides the name of a file into which to write telemetry debugging information. The default is not to write debugging information at all. If ⟨*path*⟩ is a single hyphen ("`-`"), then the log will go to the standard output.

`-telemetry-name` ⟨*string*⟩
: If the server is configured to send telemetry metrics, this provides the name of the application for purposes of identifying this running instance of the server. Defaults to "`gma-server`".

CLIENT INITIALIZATION

When a client connects, the server begins by sending a number of messages up front, before and/or after successfully authenticating. The initial negotiation with the client goes through the following stages:

Connection The server declares the protocol version it is using.

Preamble The server sends a number of messages to the client, which may include comments, declaring party members, notice of software updates, and campaign information. We recommend limiting the preamble to comments.

Authentication If configured to do so, the server will demand a valid credential from the client before proceeding any further.

Post-auth After successful authentication (or unconditionally if no authentication is required), the server sends a number of messages just as described for the preamble stage.

Ready The server then signals to the client that the negotiation is completed and the client is then free to issue any commands to the server, and may receive any messages from the server. (Before this point, the server won't even consider receiving commands from the client that aren't part of this negotiation, and won't be sending normal traffic to the client yet.)

Sync Finally, the server may send additional data to the client (typically this is synchronization data to catch the client up to the server's current notion of the game state).

By default, the preamble, post-auth, and sync stages are effectively nil. However, the presence of a client initialization file via the `-init-file` option specifies what to send to the client during negotiation.

Each line of the file is a server message to be sent to the client, formatted as documented in the server protocol specification. (I.e., a command word followed by a space and a JSON parameter object.) Long commands may be continued over multiple lines of the file, as long as the brace ("{") that begins the JSON data appears on the line with the command name, and all subsequent lines are indented by any amount of whitespace. The final brace ("}") that ends the JSON data may appear at the end of the last line or on a line by itself (in which case it need not be indented itself).

The commands which may appear in the initialization file include the following:

`//` This line is transmitted AS-IS to the client. This command does not require JSON data to follow it. All text from the "//" to the end of the line are considered a comment and are sent verbatim. This is useful to provide a human-readable message to anyone connecting to the game port.

Clients may interpret what they see in comment messages from the server but are not under any obligation to do so. Currently, the following special comment is recognized by the `mapper`(6) client (at least):

`// notice:` ⟨*message*⟩
: If the comment begins with the string "`notice:`" (not counting whitespace), then the ⟨*message*⟩ following it will be shown to the user. In this way, the GM or other server administrator may communicate urgent notices to all the users of their game server. This notice comment must appear before the `READY` command in the server's init file.

`AC`
: Add a character to the client's quick-access context menu. Typically this is the party of player characters. Any JSON parameters accepted by the server `AC` message may be given, but for the purposes of the client initialization, the important ones are `ID`, `Name`, `Color`, `Size`, `Area`, and `CreatureType`, providing a unique ID for the character, their name as it appears on the map, the color of their threat zone, creature size category, threatened area size category, and creature type (1 for monsters or 2 for players).

`DSM`
: Defines a condition status marker that may be placed on creature tokens. This will update an existing marker already known to the mapper, or add new ones to the set of condition markers. The parameters are

  `Condition`
  : The name of the condition. While this is arbitrary, it should be short, preferably a single word. It should not begin with an underscore to avoid conflicts with internal names used by the GMA software.

  `Shape`
  : Describes the shape of the marker drawn over the token. See the protocol documentation in `mapper-protocol`(7).

  `Color`
  : The color of the marker.

  `Transparent`
  : If present and true, this means to use a semi-transparent creature token when this condition is in effect.

  `Description`
  : A sentence or paragraph describing the effects of that condition.

`QOS` Sets quality of service limits in the server. If a client session violates any of these limits, its session will be terminated immediately. Make sure that whatever values you configure here are far enough out of bounds to justify ejecting the offending client. The value for this item is a JSON object where each entry is a QoS rule to enforce, and the corresponding value for the rule is a set of fields as described below. If any of these rules are not included in the `QOS` payload, that rule will not be enforced at all.

`QueryImage`
: Reject clients which excessively ask for the same images after being informed of where to find them by the server. There should be a little allowance for the client to take the time to obtain the image, so a small number of repeated requests is ok, but a properly functioning client should stop asking for the same image right away. This rule's value is a JSON object with these fields:

  `Count` *(int)*
  : The maximum number of `AI?` requests a client can send for the same image after it's already been answered by the server.

  `Window` *(duration)*
  : If this field is omitted or is blank, the client will be ejected if it ever exceeds `Count` requests for the same image after the server has answered it. Otherwise, this specifies a duration in a form such as "`15m`" or "`1h30m`" which indicates that in order to trigger the rule, `Count` repeated requests must arrive during this period of time. (The server will reset the counters every time this much time elapses.)

`MessageRate`
: Reject clients which send more than a certain number of requests during a given window of time. This rule's value is a JSON object with these fields:

  `Count` *(int)*
  : The maximum number of messages allowed for the client to send during a time `Window` before the rule is triggered.

  `Window` *(duration)*
  : Just as with the `QueryImage` rule, this specifies the time frame in which the threshold number of messages isn't allowed to be exceeded.

`Log`
: Enables a periodic logfile record of QoS metrics. If QoS debugging is enabled (`-debug qos`) or (`-debug all`), details about the data collected for each of the enabled rules is logged. Otherwise a single line is logged per client connection, in the form "`QoS img=`⟨*icount*⟩, `rate=`⟨*rcount*⟩/⟨*rmax*⟩", where ⟨*icount*⟩ is the number of different image requests currently being tracked, ⟨*rcount*⟩ is the number of packets received so far in the current time window, and ⟨*rmax*⟩ is the threshold number of packets allowed before the client is rejected.

  `Window` *(duration)*
  : The time interval after which to print each log message.

`REDIRECT`
: Instructs the client to use a different server for this session. This is used when the GM wants to use an alternate server temporarily without requiring the players to reconfigure all their clients. The JSON payload includes the following values:

  `Host`
  : The host name or IP address of the server to connect to for this session.

  `Port` *(int)*
  : The TCP port number on which to connect to the server.

  `Reason`
  : An explanation of why the redirect is being performed (optional).

  The server will disconnect from the client immediately after issuing the `REDIRECT` command to it.

`UPDATES`
: Advertises to the client the version of each software package you recommend for them to use. The JSON data has a single parameter called `Packages` which is a list of objects with the following parameters:

  `Name`
  : The name of the package, such as `mapper`, `go-gma`, or `core`.

  `MinimumVersion`
  : If a server wishes to limit clients from this package to only those with a minimum version number (as self-reported by the client in its `AUTH` message), then a

`MinimumVersion` and `VersionPattern` field must be added to that package's information here. The `MinimumVersion` field is a string with the minimum client version allowed to be used, as a semantic version string such as "`1.2`", "`1.7.3`", "`1.6-alpha.1`", etc. This will be matched against the value captured from the client's version number via the `VersionPattern` field.

`VersionPattern`
: This gives a regular expression which is matched against the `Client` field sent by the client as part of its `AUTH` message when signing on to the server. This expression MUST contain a single capturing group which yields the client's version number to be compared against the `MinimumVersion` field described above.

  See the `sample.init` file shipped with the go-gma source code for an example of this, or note that the regular expression to match the `mapper`(6) client is `"^\\s*mapper\\s+(\\S+)"`.

  Note that backslashes in the regular expression need to be escaped with another backslash (i.e., `\\`) to satisfy the encoding requirements for JSON.

`Instances`
: A list of available versions of the package. If multiple versions are listed here, they should each be for a different platform. Each instance value is an object with the following fields:

`OS`	The target operating system for this version of the package. If omitted or blank, it is OS-independent. Values are `freebsd`, `linux`, `darwin`, `windows`, etc.
`Arch`	The target hardware architecture for this version. Values are `amd64`, etc.
`Version`	The recommended version you want players to use.
`Token`	If you provide a downloadable copy of the software on your server for players to get, specify the download token here. The mapper tool currently has the capability to self-upgrade based on this token. The mapper is configured with the option `--update-url=`⟨*base*⟩ which is combined with the ⟨*token*⟩ value to get the filename to be downloaded from your server. The URLs retrieved will be ⟨*base*⟩`/`⟨*token*⟩`.tar.gz` and ⟨*base*⟩`/`⟨*token*⟩`.tar.gz.sig`.

`WORLD` Sends campaign information. The fields of the JSON payload include

`Calendar`
: Names the calendar in play.

`ClientSettings`
: Overrides some of the server- and game-specific client preference settings. The value is a JSON object with the following fields:

  `MkdirPath`
  : The path to the `mkdir` program on the server (used for GM uploads of mapper content to the server).

  `ImageBaseURL`
  : The base URL from which the client will retrieve map and image files.

  `ModuleCode`
  : The current module's ID code.

  `SCPDestination`
  : The directory where GM uploads of mapper content should be sent to.

  `ServerHostname`
  : The hostname (and optionally username in the form ⟨*name*⟩`@`⟨*host*⟩) for the GM to upload mapper content to the server.

`AUTH` This command word (without JSON data) in the initialization file causes the server to perform the authentication step before continuing. Thus, it marks the end of the preamble stage. Following lines will be sent as part of the post-auth stage.

`READY` This command word (without JSON data) in the initialization file causes the server to signal to the client that the negotiation is complete and normal client/server interaction may begin. Thus it marks the end of the post-auth stage. Anything after this point is for the sync stage.

In this final part of the file (after the `READY` command), any of the following server messages may be included to be sent to the client: `//`, `AC`, `AI`, `AI?`, `AV`, `CC`, `CLR`, `CLR@`, `CO`, `CS`, `DD=`, `DSM`, `I`, `IL`, `L`, `LS-ARC`, `LS-CIRC`, `LS-LINE`, `LS-POLY`, `LS-RECT`, `LS-SAOE`, `LS-TEXT`, `LS-TILE`, `MARK`, `OA`, `OA+`, `OA-`, `PROGRESS`, `PS`, `REDIRECT`, `ROLL`, `TB`, `TO`, `UPDATES`, or `WORLD`. (Technically, any of these commands can appear anywhere in the initialization file, but we strongly recommend limiting commands to `//`, `AC`, `DSM`, `REDIRECT`, `UPDATES`, and `WORLD` in all stages except the final (sync) stage.)

`SYNC` This command word (without JSON data) in the initialization file will cause the server to behave as if the client sent a `SYNC` command to it after the negotiation is complete. This sends the full game state to the client, so that a newly connected mapper will display the current map contents the other players see.

SECURITY

The authentication system employed here is simplistic and not ideal for general use, but is considered to be good enough for our purposes here, since the stakes are so low. It is intended just to discourage cheating at the game by looking at spoilers or direct messages intended for other users, not for any more rigorous protection.

The challenge/response system employed by the server is designed to resist replay attacks since it does not divulge the actual password in the clear over the network, although other attacks such as man-in-the-middle remain possible. This authentication mechanism is used because at this point the server and clients do not support encrypted communications. (If this becomes supported in the future, a more robust authentication mechanism will be possible which does not have the weaknesses documented here.)

The main weakness of the system is that passwords are stored in plaintext on the server and on each client, which means it is critical to secure the password file and the system itself. Caution your players to use a password for the mapper that is different from any other passwords they use (which should be the password practice people observe anyway). A breach that reveals passwords from the server's file, or the client configuration files where passwords are stored, would then only allow an imposter to log in to your map service, which admittedly is more of an inconvenience than a serious security issue, assuming you use your map server just for playing a game and not for the communication of any sensitive information.

Don't use the GMA mapper server for the communication of sensitive information. It's part of a game. Just play a game with it.

SIGNALS

The map service responds to the following signals while running. These actions may not be taken immediately but should happen within a few seconds.

`HUP` This signal terminates all existing client connections but leaves the server up and ready to accept new incoming connections.

`INT` Gracefully shuts down the server.

`USR1` Causes the server to re-read its initialization file. Clients which connect after this will see the new initialization information. This also jumps the next message ID for chat messages and die roll results to most likely be a larger ID than other servers (it sets the next ID to the current UNIX timestamp value, just as the server does when it starts; this will make it ahead of other servers on the assumption that server clocks are correct and no server will sustain a message rate of >= one message per second since it was started).

`USR2` This signal causes the server to dump a human-readable description of the current game state database to the log file.

SEE ALSO

`gma`(6), `gma-mapper`(5), `gma-mapper`(6).
The server communications protocol is definitively documented in the `gma-mapper`(6) manpage which comes with the GMA-Mapper package.

AUTHOR

Steve Willoughby / steve@madscience.zone.

BUGS

If the server is not configured to require authentication, that means it won't drop nuisance connections either, since it's accepting all connections as valid, even if it never sends any valid data to the server.

gma-go-session-stats(6)

NAME

gma go session-stats – Generate web forum summaries of game sessions

SYNOPSIS

(If using the full GMA core tool suite)

`gma go session-stats` [options as described below...]

(Otherwise)

`session-stats -h`

`session-stats -help`

`session-stats` [`-r`] `-s` ⟨*inputfile*⟩

`session-stats -v` ⟨*inputfile*⟩

DESCRIPTION

When completing a game session that is part of a larger overall campaign, the GM can collect some basic statistics about the game session in a JSON-formatted data file. This has the overall format:

```
{"game_sessions": [
   ⟨each game session described in a list here...⟩
]}
```

Each game session is described by a JSON object in the form

```
{
   "date": "18-Feb-2023",
   "video": "ABCDEFGHIJKL",
   "duration": "5h56m34s",
   "title": "The Plot Thickens...",
   "world_dates": "Desnus 4, 4722",
   "book": 1,
   "url": "https://forum.example.org/games/1.html",
   "synopsis": "The party stumbles upon a dark hidden secret."
}
```

Note that, to be a valid JSON file, the trailing comma must be omitted from the last game entry in the list.

The meaning of the fields are described below.

`date`
: The real-world date of the game session, in ⟨*dd*⟩-⟨*month*⟩-⟨*yyyy*⟩ format.

`video`
: If your game sessions are recorded on YouTube, the video ID (the part of the video's URL that appears just after the "`v=`" but before the next "`&`") is entered here. If you don't have a video, leave this field out.

`duration`
: The total duration of the session (or of the edited video, if there is one) in Go's duration type format.

`title`
: The title by which you wish to refer to this game session.

`world_dates`
: The in-game date(s) in which the events of the game session take place. This is an arbitrary-formatted string value.

`book`
: The integer book number of a multi-book campaign or Adventure Path. The synopsis list will indicate the start of each new book in the list.

`url`
: The URL to the web forum post where you have written a full summary of the game session events.

`synopsis`
: A string value which contains an HTML-formatted brief synopsis of the events of the game session.

With this information in place, `session-stats` provides a way to generate two different summaries of the session information which you can post to your game forum site: a synopsis list and a video/session list.

Synopsis List

By running `session-stats` with a `-s` option, it produces on its standard output an HTML-formatted list of game session synopses, with links to the full description entries in your forum. The HTML output is bracked inside "`[html]`" and "`[/html]`" BBC tags to facilitate its use in a BBC-enabled web forum. Ignore these tags in a pure HTML environment.

Video/Session List

By running `session-stats` with a `-v` option, it produces on its standard output an HTML-formatted list of game sessions and video links. The HTML output is bracked inside "`[html]`" and "`[/html]`" BBC tags to facilitate its use in a BBC-enabled web forum. Ignore these tags in a pure HTML environment.

OPTIONS

`-h`, `-help`
: Print a summary of options and exit.

`-r`
: Reverse the order of the output produced by `-s` so that it reads from most recent to earliest game session.

`-s`
: Produce the list of game synopses.

`-v`
: Produce the list of session video links.

SEE ALSO

`gma`(6).

AUTHOR

Steve Willoughby / steve@madscience.zone.

BUGS

gma-go-upload-presets(6)

NAME

gma go upload-presets – Upload die-roll presets to a GMA server

SYNOPSIS

(If using the full GMA core tool suite)

`gma go upload-presets` [⟨*args*⟩ ...]

(Otherwise)

`upload-presets -endpoint` [⟨*hostname*⟩]:⟨*port*⟩ [`-for` ⟨*username*⟩] [`-pass` ⟨*password*⟩] [`-replace`] [`-user` ⟨*username*⟩] ⟨*file*⟩ ...

DESCRIPTION

`Upload-presets` connects to a GMA server and uploads the die-roll presets from one or more files, so they are stored in the server for use during game play.

The files must be in the format documented by `gma-dice`(5), in either format version 1 or 2.

OPTIONS

The following options control the action of `upload-presets`.

`-endpoint` [⟨*hostname*⟩]:⟨*port*⟩
: Connect to the server at the specified TCP port.

`-for` ⟨*username*⟩
: Store the presets for ⟨*username*⟩ instead of the user you are logged in as.

`-pass` ⟨*password*⟩
: Log in to the server with the specified ⟨*password*⟩

`-replace`
: Replace the user's entire set of presets with the uploaded data. By default the uploaded presets will be added to any that already exist for that user.

`-user` ⟨*username*⟩
: Log in to the server with the specified ⟨*username*⟩ (default "`GM`").

SEE ALSO

`gma`(6), `gma-mapper`(5), `gma-mapper`(6), `gma-go-server`(6).

AUTHOR

Steve Willoughby / steve@madscience.zone.



Protocol Definitions

> Anyone who considers protocol unimportant has never dealt with a cat.
>
> —Robert A. Heinlein, *The Cat Who Walks Through Walls*

Traditionally, section 7 of the Unix manual, whose entries thus end with the suffix "(7)", is used for miscellaneous documents, including the definition of networking protocols. For GMA, we have the specification of the protocol used between GMA servers and their clients (typically the GMA Mapper client) as well as documentation for markup syntax, die rolls, and PostScript preambles.

form-preamble(7)

NAME

form-preamble.ps – Generic PostScript Form Preamble

DESCRIPTION

This is a generic form-building PostScript preamble we use for applications which need to generate forms and tabular data. It defines the functions documented here. The GMA-specific PostScript preamble `gma-preamble`(7) assumes `form-preamble` was already loaded before it.

The operators defined here represent a low-level API for rendering form content in PostScript. See `gma-preamble` for a higher-level application-specific API built on top of this one.

PostScript Operator Notation

The documentation that follows uses the standard PostScript conventions for illustrating operators. The list of input values which are to be pushed onto the stack are listed first (or `--` if no input parameters are consumed by the operator), then the operator name itself, then the list of output values which the operator will push on the stack (or `--` if the operator doesn't push any values).

Shorthand Aliases

The following operators are defined as convenient short-hand aliases for other operators:

`-- cp --`
: Alias for `closepath`.

`-- ln --`
: Alias for `lineto`.

`-- mv --`
: Alias for `moveto`.

`-- np --`
: Alias for `newpath`.

`-- ok --`
: Alias for `stroke`.

`-- rln --`
: Alias for `rlineto`.

Movement Operators

The current position is tracked by variables `X` and `Y`. The following operators manipulate this current position.

⟨*h*⟩ `D --`
: Move down by ⟨*h*⟩ points.

⟨*w*⟩ `R --`
: Move right by ⟨*w*⟩ points.

⟨*h*⟩ `RequiredVerticalSpace --`
: Ensure ⟨*h*⟩ points of vertical space exists below the current position. If that much space is not available, eject the current page and start a new one.

Form-Drawing Operators

⟨*x*⟩ ⟨*y*⟩ ⟨*w*⟩ ⟨*h*⟩ `BoxFill --`
: Draw a filled box with (⟨*x*⟩, ⟨*y*⟩) at the top-left corner, with width ⟨*w*⟩ and height ⟨*h*⟩.

⟨*x*⟩ ⟨*y*⟩ ⟨*w*⟩ ⟨*h*⟩ `BoxFrame --`
: Draw an empty, unfilled box with (⟨*x*⟩, ⟨*y*⟩) at the top-left corner, with width ⟨*w*⟩ and height ⟨*h*⟩.

⟨*x*⟩ ⟨*y*⟩ ⟨*w*⟩ ⟨*h*⟩ `BoxPath` ⟨*path*⟩
: As `BoxFrame`, but leaves the box's path on the stack instead of drawing it.

⟨*w*⟩ ⟨*h*⟩ `RelativeBoxFill --`
: As `BoxFill`, but draws the box relative to the PostScript interpreter's current point.

⟨*w*⟩ ⟨*h*⟩ `RelativeBoxFrame --`
: As `BoxFrame`, but draws the box relative to the PostScript interpreter's current point.

⟨*w*⟩ ⟨*h*⟩ `RelativeBoxPath` ⟨*path*⟩
: As `BoxPath`, but the path is constructed with the given width and height from the current point of the PostScript interpreter.

Color and Theme Management Operators

`-- AltGreyHue --`
: Set the fill color for an alternative "greyed-out" entry box.

`-- BlankHue --`
: Set the fill color for a blank entry box.

`-- DarkHue --`
Set the fill color for a darkly "greyed-out" entry box.

`-- FormHue --`
Set the line and text color for the general form elements.

`-- GreyHue --`
Set the fill color for a "greyed-out" entry box.

`-- LabelTagHue --`
Set the text color for box labels.

`-- SetColor_data --`
Switch to the standard form color for data and running text.

`-- SetColor_form --`
Switch to the standard form color for borders, non-data text, etc.

`-- SetLine_thin --`
Set PostScript's line width to a "thin" size.

`-- SetLine_med --`
Set PostScript's line width to a "medium" size.

`-- SetLine_wide --`
Set PostScript's line width to a "wide" size.

Font Management Operators

`-- FontFace_body` ⟨*font*⟩
Returns the font to be used for main body text.

`-- FontFace_data_ttl` ⟨*font*⟩
Returns the font to be used for data titles.

`-- FontFace_footnote` ⟨*font*⟩
Returns the font to be used for footnotes.

`-- FontLead_body` ⟨*size*⟩
Returns the vertical distance in points between baselines.

`-- FontSize_body` ⟨*size*⟩
Returns the size in points to use for the body font.

`-- FontSize_data_ttl` ⟨*size*⟩
Returns the size in points to use for the data titles.

`-- FontSize_footnote` ⟨*size*⟩
Returns the size in points to use for footnotes.

`-- SelectBodyFont --`
Select the body font (`FontFace_body` at `FontSize_body` points).

`-- SelectDataTitleFont --`
: Select the data title font (`FontFace_data_ttl` at `FontSize_data_ttl` points).

`-- SelectFootnoteFont --`
: Select the footnote font (`FontFace_footnote` at `FontSize_footnote` points).

Housekeeping Operators

`-- appEject --`
: The application using this preamble should re-define this operator. It will be called at the end of each page.

`-- appStart --`
: The application using this preamble should re-define this operator. It will be called at the start of each page of output. N.B. it may also be called at the end of the job, to set up for the next page that doesn't actually get created.

`-- eject --`
: Eject the current page and start a new one.

Text Typesetting Operators

⟨*string*⟩ ⟨*font*⟩ ⟨*size*⟩ ⟨*w*⟩ ⟨*x*⟩ ⟨*y*⟩ `FitText --`
: Typeset ⟨*string*⟩ in the given ⟨*font*⟩, preferably at the specified point ⟨*size*⟩. The text is drawn on a baseline at the given (⟨*x*⟩, ⟨*y*⟩) position and must fit within a horizontal space ⟨*w*⟩ points wide. If the string won't fit, the point size is reduced until it does fit.

⟨*string*⟩ ⟨*font*⟩ ⟨*size*⟩ ⟨*w*⟩ ⟨*x*⟩ ⟨*y*⟩ `FitTextCtr --`
: As `FitText`, but the ⟨*string*⟩ is centered within the designated space.

⟨*string*⟩ ⟨*font*⟩ ⟨*size*⟩ ⟨*w*⟩ ⟨*x*⟩ ⟨*y*⟩ `FitTextR --`
: As `FitText`, but the ⟨*string*⟩ is right-justified within the designated space.

⟨*string*⟩ ⟨*font*⟩ ⟨*size*⟩ ⟨*w*⟩ `RelativeFitText --`
: Like `FitText`, but the starting location for the text is the interpreter's current point.

⟨*string*⟩ ⟨*x*⟩ ⟨*y*⟩ `RenderText` ⟨*w*⟩ ⟨*h*⟩
: Draw ⟨*string*⟩ in the current font on a baseline at the given point. Returns the text dimensions as rendered.

⟨*x*⟩ ⟨*y*⟩ ⟨*lead*⟩ `WaF_init --`
: Set the starting position and interline spacing for `WrapAndFormat` (you must call this first before calling `WrapAndFormat` for the first time per text block).

⟨*array*⟩ ⟨*w*⟩ ⟨*showproc*⟩ ⟨*nlproc*⟩ `WrapAndFormat --`
: Given an array of "chunks" of running text, typesets the text starting at the current point of the PostScript interpreter. If a chunk would put the text block wider than ⟨*w*⟩, a new line is started for that chunk instead.

  The ⟨*nlproc*⟩ procedure is invoked every time we need to move to a new line. The ⟨*showproc*⟩ procedure is likewise called to actually display each chunk. Procedures `WaF_nl` and `WaF_show` are provided which may be used for ⟨*nlproc*⟩ and ⟨*showproc*⟩ respectively.

  Each chunk is itself an array of 3 elements: `[` ⟨*eproc*⟩ `[`⟨*s0*⟩ ⟨*s1*⟩ ... ⟨*sN*⟩`]` ⟨*sproc*⟩ `]`; ⟨*sproc*⟩ is called to set up the chunk (selecting fonts, etc.), then each string ⟨*s0*⟩, ⟨*s1*⟩, and so on are typeset, with new lines started as necessary between each of those strings. Then ⟨*eproc*⟩ is called after the chunk is typeset. It is legal for ⟨*sproc*⟩ and ⟨*eproc*⟩ to directly call ⟨*showproc*⟩ or ⟨*nlproc*⟩ themselves.

Form Typesetting Operators

⟨*font*⟩ ⟨*size*⟩ ⟨*h*⟩ `BeginDataBlock --`
: Start a block of form cells at the current (`X`, `Y`) location. The data to be written in each cell will be in the specified ⟨*font*⟩ and ⟨*size*⟩ (but the size of any given cell may be reduced to get the data to fit).

`-- EndDataBlock --`
: Ends a row of data cells begun by `BeginDataBlock`. This starts a new line on the form.

⟨*data*⟩ ⟨*title*⟩ ⟨*w*⟩ `DataBlock --`
: Typesets a cell in a data block row. You must call `StartDataBlock` before calling `DataBlock` for each cell in the row. This typesets ⟨*data*⟩ inside a box of the given width ⟨*w*⟩ with a smaller-sized ⟨*title*⟩ written at the very top of the box. You may call a hue-changing command before invoking `DataBlock` to alter the color used to draw and fill the box.

⟨*data*⟩ ⟨*bg*⟩ ⟨*title*⟩ ⟨*w*⟩ `DataBlockCR --`
: As above, but right-justifies ⟨*data*⟩ in the box. Additionally, ⟨*bg*⟩ will be written in white lettering underneath the ⟨*data*⟩ string (so it is expected that the box itself be shaded).

⟨*data*⟩ ⟨*bg*⟩ ⟨*title*⟩ ⟨*w*⟩ `DataBlockLR --`
: As `DataBlockCR`, but ⟨*bg*⟩ is centered in the box.

⟨*data*⟩ ⟨*title*⟩ ⟨*w*⟩ `DataBlockCtr --`
: As `DataBlock`, but ⟨*data*⟩ is centered inside the box.

⟨*data*⟩ ⟨*title*⟩ ⟨*w*⟩ `DataBlockR --`
: As `DataBlock`, but ⟨*data*⟩ is right-justified in the box.

⟨*data*⟩ ⟨*w*⟩ `TitleBlockCtr --`
Writes ⟨*data*⟩ in white, centered in a solid form-hue box of width ⟨*w*⟩.

⟨*font*⟩ ⟨*size*⟩ `ChangeDataBlockFont --`
Change the current font used for printing text in data block cells. Normally you don't need to call this since `BeginDataRow` calls it for you.

⟨*qty*⟩ ⟨*us*⟩ ⟨*chk*⟩ ⟨*hf*⟩ ⟨*bpr*⟩ ⟨*boxx*⟩ ⟨*boxy*⟩ ⟨*w*⟩ ⟨*h*⟩ ⟨*gapx*⟩ ⟨*gapy*⟩ ⟨*si*⟩ ⟨*schk*⟩ ⟨*shf*⟩ ⟨*diff*⟩ `CheckBoxMatrix -`

Draw a row of ⟨*qty*⟩ checkboxes with the first box located at (⟨*boxx*⟩, ⟨*boxy*⟩). Each box is ⟨*w*⟩ points wide and ⟨*h*⟩ points high, with ⟨*gapx*⟩ points horizontally and ⟨*gapy*⟩ points vertically between boxes. There will be at most ⟨*bpr*⟩ boxes per row (creating additional rows as necessary).

Of those boxes, ⟨*us*⟩ of them are "in play" and are drawn with solid lines. The remaining (unused) boxes are drawn with dashed lines. ⟨*Chk*⟩ is the number of boxes to draw as fully checked (with an "X" through them), while ⟨*hf*⟩ is the number of boxes to draw as half-checked (with a slash through them).

The ⟨*si*⟩ value indicates the number of consecutive boxes to shade. That many boxes will be drawn without shading, then that many with shading, and then repeat that pattern. If ⟨*diff*⟩ is true, there is a differential between shaded and non-shaded boxes (e.g., using shaded boxes for domain spell slots and unshaded ones for normal slots). In this case, we won't extend the check marks into the shaded boxes, but will rather fully check ⟨*schk*⟩ of the shaded boxes and half-check ⟨*shalf*⟩ of them.

⟨*qty*⟩ ⟨*us*⟩ ⟨*chk*⟩ ⟨*hf*⟩ ⟨*bpr*⟩ ⟨*boxx*⟩ ⟨*boxy*⟩ ⟨*w*⟩ ⟨*h*⟩ ⟨*gapx*⟩ ⟨*gapy*⟩ ⟨*si*⟩ ⟨*schk*⟩ ⟨*shf*⟩ ⟨*diff*⟩ ⟨*arr*⟩ `LabelledChe`

Like `CheckBoxMatrix` but also accepts an ⟨*array*⟩ of single-character text labels to be written into the boxes.

⟨*qty*⟩ ⟨*us*⟩ ⟨*chk*⟩ ⟨*hf*⟩ ⟨*bpr*⟩ ⟨*boxx*⟩ ⟨*boxy*⟩ ⟨*w*⟩ ⟨*h*⟩ ⟨*gapx*⟩ ⟨*gapy*⟩ ⟨*si*⟩ ⟨*schk*⟩ ⟨*shf*⟩ ⟨*diff*⟩ ⟨*title*⟩ `TitledChec`

Like `CheckBoxMatrix` but also accepts a ⟨*title*⟩ string that will be printed before the boxes.

... ⟨*minboxx*⟩ ⟨*maxboxx*⟩ ⟨*nlfunc*⟩ `*CheckBoxMatrixBounded --`
Variant forms of the `CheckBoxMatrix`, `LabelledCheckBoxMatrix`, and `TitledCheckBoxMatrix` operators, in this case called as `CheckBoxMatrixBounded`, `LabelledCheckBoxMatrixBounded`, and `TitledCheckBoxMatrixBounded` respectively, take two additional parameters on the top of the stack: ⟨*minboxx*⟩ and ⟨*maxboxx*⟩. If all the requested boxes cannot fit before reaching ⟨*maxboxx*⟩, a new row will be started at ⟨*minboxx*⟩. In order to do this, ⟨*nlproc*⟩ is called to finish tyesetting the

current row and begin the next. Before calling ⟨*nlproc*⟩, the variables `CBM__x` and `CBM__y` are updated to the location expected for the next box to be drawn, but ⟨*nlproc*⟩ may reassign values to these varaibles as desired.

SEE ALSO

`gma-preamble`(7).

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

COPYRGHT

Part of the GMA software suite, copyright © 1992–2025 by Steven L. Willoughby, Aloha, Oregon, USA. All Rights Reserved. Distributed under BSD-3-Clause License.

gma-dice-syntax(7)

NAME

gma-dice-syntax – Die-roll expressions for the GMA die roller facility

DESCRIPTION

The GMA game system includes the functionality to roll dice to get random numbers for use in the game. These are provided in programs such as `gma-mapper`(6), `gma-roll`(6), and `gma-go-roll`(6). The following document describes the various ways you can express a die roll for the system to make.

DIE-ROLL EXPRESSION SYNTAX

The general form for die roll expressions is: [⟨*name*⟩=] [⟨*qty*⟩[/⟨*div*⟩]]`d`⟨*sides*⟩ [`best`|`worst of` ⟨*n*⟩] [...] [|⟨*modifiers*⟩...]

This will roll ⟨*qty*⟩ dice, each of which has the specified number of ⟨*sides*⟩, (i.e., each generates a number between 1 and ⟨*sides*⟩, inclusive.) The result is divided by ⟨*div*⟩ (but in no case will the result be less than 1). Finally, any ⟨*bonus*⟩ (positive or negative) is added to the result. If ⟨*factor*⟩ is specified, the final result is multiplied by that amount.

As a special case, ⟨*sides*⟩ may be the character "%", which means to roll percentile (d100) dice, as explained below.

Basics

The basic die-roll expression uses traditional dice notation (e.g., "`3d6`" means to roll three six-sided dice and add them together). Multiple such dice may appear, separated from each other (and from numeric constants) with the basic math operators + (for addition), - (for subtraction), * (for multiplication), or // (for division; note that is two slashes); you can also use Unicode characters U+00D7 (×) for multiplication and U+00F7 (÷) for division. The standard algebraic order of operations is performed, and parentheses can be used as necessary to force a particular order. Thus, "`d20+17`", "`4d8+3d10+27`", and "`(2d10+3)*3`" are all valid dice rolls.

Constraints

The operators <= and >= indicate that the value to their left must be less than or equal to (or greater than or equal to) some maximum (or minimum) value. For example "`3d6<=10`" means "roll 3d6 but the result must be less than or equal to 10"; if the roll yielded a result greater than 10, 10 will be used instead. Likewise, "`2d10>=5`" rolls 2d10 but forces the result to be at least 5.

Since these apply to the value immediately to their left, parentheses are needed if you want to place the constraint on a larger expression, such as "`(3d6+27+1d10)>=10`".

Unicode U+2264 and U+2265 ($\leq$ and $\geq$) may also be used instead of <= and >= respectively.

Special

If a dice value is prefixed with a > symbol, as in "`>5d10`", then the first die will be assumed to come up with its maximum value (so what's really rolled in this example is `10+4d10`).

Rerolls

If you put "`best of` ⟨*n*⟩" or "`worst of` ⟨*n*⟩" after a dice value, such as "`d20 best of 2`", it will roll the die that may times and take the best (or worst) of all those rolls to use for that set of dice. Note that this is part of the dice, not the overall expression so you would say, for example, "`3d6 best of 3 + 12`" and not "`3d6+12 best of 3`".

Percentile Rolls

You can use "`d%`" to roll a d100 or "percentile" die as part of any die-roll expression. To just get a roll to indicate something that is successful ⟨*x*⟩ percent of the time, the special die-roll form "⟨*x*⟩`%`" can be used (this cannot be part of a larger expression). For example, if your die roll was "`42%`", then the result reported back would be 1 (success) with a 42% probability, or 0 (failure) with a 58% probability.

To customize the success/failure messages, you can add those messages to the end, like "`42% miss`" (where the result will be "miss" 42% of the time or else "hit"), or "`15% red/blue`" (where the result will be "red" 15% of the time or else "blue").

Titles and Labels

You can put arbitrary text at the start of the string, followed by an equals sign (=), to place a title on the whole roll, like "`Attack roll = d20+12`".

Likewise, any dice or constant values in the expression can be followed by a text label to indicate what that value represents. In this case, the label must consist of one or more words which begin with a letter or underscore (_), and contain only letters, digits, underscores, periods (.), and commas (,). For example, "`5d6 + 1d6 fire + 1d6 acid spray`".

Custom Colors for Titles and Labels

To make a die roll title stand out, for example to make damage monster attack rolls or critical damage rolls stand out so you can notice them at a

glance when scrolling back through the die-roll/chat window, you can colorize the title by adding color codes after the title text (just before the equals sign (=). The color code is introduced by a Unicode character U+2261 (≡), followed by the name of a color or a pound sign and a hex RGB color value such as ≡`red` or ≡`#3366bb`. If you add two of these color codes, the first sets the text foreground color and the second sets the background color. For example, "`damage≡red≡white=...`" titles a die roll with the text "damage" in red letters on a white background.

You can break a title into multiple parts, each of which may have its own coloring by separating the parts with Unicode characters U+2016 (`||`). For example, "`beholder≡red||damage≡yellow=2d6+12`".

While text labels in the die-roll expression itself (e.g., the "fire" in "6d6 fire") are normally displayed according to your overall style preferences (so all such labels are displayed the same), sometimes die roll expressions can be so complex, and a few key modifiers so special, that it may be important for certain labels to visually stand out when they appear. To accommodate these cases, it is possible to apply the same color codes to the end of these labels as described above for the die-roll titles. We recommend doing this sparingly, however, so that they remain an effective way of highlighting exceptional modifiers rather than just introducing an explosion of color into the die-roll window.

For example: "`d20+15 bab+3 str+4 smite≡red+2 luck`".

Normally, these colors apply to the entire text label which may include spaces. If you just want to color part of the label, or want to apply different colors to different parts of the label, use the U+2016 double-vertical-bar separator (`||`) between parts of the label to divide portions of the modifier label to be colorized, similar to how this is done for the overall die-roll title string.

Note that not all clients support displaying colors like this, but the server will remove the color codes before sending die-roll results to them so there's no harm including them in your die rolls. Clients which don't support them won't need to worry about them at all.

Options

At the very end of the entire die-roll string, you can place one or more of the following options which alter the meaning of the whole thing. Each option begins with a vertical bar (`|`) character. Spaces between option names and their numeric arguments are optional.

`|max` ⟨*x*⟩
: Force the whole overall result to be no more than ⟨*x*⟩.

`|min` ⟨*x*⟩
: Force the whole overall result to be at least ⟨*x*⟩.

`|c`
: This is an attack roll. If the die rolled a natural 20, add a second roll to confirm the critical hit.

`|c` ⟨*t*⟩
: As `|c` but consider a natural roll of at least ⟨*t*⟩ to threaten the critical.

`|c` ⟨*t*⟩±⟨*b*⟩
: As above but add the bonus ⟨*b*⟩ to the confirmation roll.

`|dc` ⟨*n*⟩
: Indicate that the roll was a "success" if the result was at least ⟨*n*⟩.

`|maximized`
: All die rolls are forced to their maximum possible values.

`|repeat` ⟨*n*⟩
: Roll ⟨*n*⟩ times.

`|sf`
: This roll is subject to automatic success/fail rules: the roll is successful if the die rolled a natural 20 (or whatever the die's maximum is) and failed if a natural 1 was rolled.

`|sf` ⟨*success*⟩[/⟨*fail*⟩]
: As `|sf` but specify your own custom messages to print if successful (and failure if that is given as well, separated by a slash).

`|total` ⟨*n*⟩
: Continue rolling until the total cumulative result is at least ⟨*n*⟩.

`|until` ⟨*n*⟩
: Continue rolling until the result is at least ⟨*n*⟩.

Permutation

If a slash-separated set of values appear between curly braces, the die-roll will be executed once for each of the values in turn. For example, the die roll "`d20+{17/12/7/2}+5`" is the same as making the four separate rolls "`d20+17+5`", "`d20+12+5`", "`d20+7+5`", and "`d20+2+5`".

Likewise, the die roll "`d20+{10/8}+{d10/100}`" is the same as rolling "`d20+10+d10`", "`d20+10+100`", "`d20+8+d10`", and "`d20+8+100`".

Any arbitrary text may appear in the braces; the permutations and substitutions are made before each resulting die-roll string is parsed and evaluated.

The order in which the permuted rolls are performed is not defined.

CLIENT-DEPENDENT FEATURES

There are some additional features that are supported by the server but actually implemented client-side so their availability will depend on the actual client you use. We'll touch on them briefly here but see the documentation for your client for more details.

Presets and Modifiers

Clients can store pre-set die rolls for common rolls you make so they are prepared ahead of time so you don't have to type them every time, and can instead just invoke them with a single click or other easy action. They may be stored locally in a disk file or stored in the game server for you.

This may include full die rolls for things like your usual weapon attacks, or common modifiers to add to other die rolls which you may want to selectively turn on or off as they apply to whatever you're rolling at the time.

Variables

Some preset modifiers you store in your client may be assigned names, as variables which you can then place into larger die-roll expressions. For example, if you have a common bonus where you add +1d6 fire damage to certain attacks, you can store that into a preset variable called `firedmg` and then add it to all the rolls where it applies by putting a dollar sign in front of its name, like

```
damage roll = d6+5+$firedmg
```

Assuming that you had defined the value "`1d6 fire`" into the `firedmg` variable, then the actual die roll sent to the server would be "`damage roll = d6+5+1d6 fire`".

If you need to be careful about confusing the name of the variable with surrounding text in the die roll, you can enclose its name in curly braces like this: `${firedmg}`. Also, your GM may define system-wide presets that are automatically available for everyone to use, including named variables. In that case, you can refer to them using double dollar signs, as in `$$firedmg` or `$${firedmg}`.

Lookup Tables

Some clients also support random action lookup tables. In this case there is a game rule which says you roll dice and consult a table to see what action to take or what the outcome of a magical effect is. You can encode that table into a preset in your client and store it under a name just as for a named variable.

For example, if your character has a magic item called a *wand of wonder* and you use it often but don't want to have to look up the random effect on the table of 22 different possible outcomes all the time. So you create a

lookup table in your client's presets called, say, `wonder` which includes the fact that you need to roll a d100 and what all the possible outcomes should be based on what the result of that roll turns out to be. Now, in your die roller client you can invoke the table lookup by simply typing

```
#wonder
```

(along with other modifiers or options if you need them). The client will send the needed die roll (d100 in this case) and then look up the result in the table for you. If, for example, it happend to roll 8, it would look that up and see that the result is that the target is surrounded by *faerie fire* so the client will send out a message to that effect to the other players on the chat channel.

Just as with variables, you can surround the name in braces and use double pound-signs for tables defined at the system-wide level by the `GM`, as in `#{wonder}`, `##wonder`, and `##{wonder}`.

SEE ALSO

`gma-roll`(6), `gma-go-roll`(6).

NOTES

Why `$` for variable names and `#` for table names? Just because there's a long standing tradition of using the former as a variable prefix in scripting languages such as Unix and Linux shell scripts and scripting languages such as perl, awk, and tcl. In the latter case, because the octothorpe or pound sign visually resembles the horizontal and vertical rules that separate the rows and columns of a table which struck my imagination at the time and I just ran with that idea.

AUTHOR

Steve Willoughby / steve@madscience.zone.

gma-mapper-protocol(7)

NAME

gma-mapper-protocol – GMA client/server protocol specification

EXPERIMENTAL

This document includes experimental changes to the protocol which have not yet been officially released.

DESCRIPTION

The `mapper`(6) 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). To take advantage of its full multi-user capabilities, multiple `mapper` clients may be connected to the GMA `server`(6). The following document is the definitive specification describing the protocol used to communicate all messages between these programs (as well as others which interact with the GMA server, such as `gma-console`(6)).

When a client is connected to a GMA server, they exchange commands to indicate changes to the map display or other game state information. 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 key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

The software implementing this protocol 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 sent which were not expected by the receiver 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 419. 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, `fixed-width` 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`, `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. (Note that prior to protocol 414, servers were not allowed to send `DENIED` to clients after the `READY` message. Now they can send a `DENIED` message at any time to indicate that they wish to terminate the client's session immediately.)

Command Summary

The messages sent by clients and servers is summarized in Tables F.1 and F.2, and then explained in detail in the following paragraphs. In the table, 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.

Command	C↔S	Reply	Priv	Data	Description
`//`	C↔S	—	no	any	Human-readable comment
`/CONN`	C→S	`CONN`	no	none	Request list of peer clients
`ACCEPT`	C→S	—	no	JSON	Subscribe to a set of messages
`AC`	C←S	—	N/A	JSON	Add primary PC to party
`ALLOW`	C→S	—	no	JSON	Indicate supported features
`AUTH`	C→S	`GRANTED`	no	JSON	Authenticate to server
`AV`	C↔S	—	no	JSON	Scroll clients' views
`AI`	C↔S	—	no	JSON	Add image to client's known set
`AI?`	C↔S	—	no	JSON	Ask for definition of image
`AI/`	C→S	—	yes	JSON	Filter saved image definitions
`CC`	C↔S	—	no	JSON	Clear the chat history
`CLR`	C↔S	—	no	JSON	Remove objects from client
`CLR@`	C↔S	—	no	JSON	Remove elements from a map file
`CO`	C↔S	—	yes	JSON	Turn on/off combat mode
`CONN`	C←S	—	N/A	JSON	Notify of all connected peers
`CORE`	C→S	`CORE=`	no	JSON	Query the core game database
`CORE=`	C←S	—	N/A	JSON	Core game data from the game database
`CORE/`	C→S	—	yes	JSON	Filter availability of the game database
`COREIDX`	C→S	`COREIDX=`	no	JSON	Get core database index
`COREIDX=`	C←S	—	N/A	JSON	Update core database index info
`CS`	C↔S	—	yes	JSON	Change the current time of day
`D`	C→S	`ROLL`	no	JSON	Initiate a die roll
`DD`	C←S	`DD=`	maybe	JSON	Store new presets for user
`DD+`	C→S	`DD=`	maybe	JSON	Add presets to user's set
`DD/`	C→S	`DD=`	maybe	JSON	Filter user's presets
`DD=`	C←S	—	N/A	JSON	Receive user's presets
`DDD`	C→S	`DD=`	maybe	JSON	Define preset delegates
`DENIED`	C←S	—	N/A	JSON	Server denies access
`DR`	C→S	`DD=`	maybe	JSON	Request user's presets
`DSM`	C↔S	—	yes	JSON	Define creature status marker
`ECHO`	C↔S	`ECHO`	no	JSON	Send data for server to echo back
`FAILED`	C↔S	—	yes	JSON	Request failed or denied by GM
`GRANTED`	C←S	—	N/A	JSON	Server grants access
`HPACK`	C↔S	—	yes	JSON	Acknowledge hit point request
`HPREQ`	C↔S	`HPACK/FAILED`	no	JSON	Request hit point change
`I`	C↔S	—	yes	JSON	Indicate turn in combat
`IL`	C↔S	—	yes	JSON	Update the initiative list
`L`	C↔S	—	no	JSON	Load elements from file
`LS-ARC`	C↔S	—	no	JSON	Add an arc to the map
`LS-CIRC`	C↔S	—	no	JSON	Add an ellipse to the map
`LS-LINE`	C↔S	—	no	JSON	Add a line to the map
`LS-POLY`	C↔S	—	no	JSON	Add a polygon to the map
`LS-RECT`	C↔S	—	no	JSON	Add a rectangle to the map
`LS-SAOE`	C↔S	—	no	JSON	Add an area of effect to the map
`LS-TEXT`	C↔S	—	no	JSON	Add some text to the map
`LS-TILE`	C↔S	—	no	JSON	Add a graphic tile to the map
`MARCO`	C←S	`POLO`	N/A	none	Check if client is alive

Table F.1: Summary of Client/Server Messages

Command	C↔S	Reply	Priv	Data	Description
MARK	C↔S	—	no	JSON	Visually mark a spot
POLO	C→S	—	no	none	Acknowledge server's ping
PRIV	C←S	—	N/A	JSON	Privileged command denied
PROGRESS	C↔S	—	no	JSON	Indicate progress on task
PROTOCOL	C←S	—	N/A	int	Signal protocol version in use
PS	C↔S	—	no	JSON	Place a creature token
READY	C←S	—	N/A	none	Server sign-on complete
REDIRECT	C←S	—	N/A	JSON	Instruct client to use a different server
ROLL	C←S	—	N/A	JSON	Result of a die roll
OA	C↔S	—	no	JSON	Update object attributes
OA+	C↔S	—	no	JSON	Add values to an attribute
OA-	C↔S	—	no	JSON	Remove values to an attribute
OK	C←S	AUTH	N/A	JSON	Ask client to authenticate
SYNC	C→S	any	no	none	Request replay of commands
SYNC-CHAT	C→S	CC/ROLL/TO	no	JSON	Request replay of messages
TB	C↔S	—	yes	JSON	Turn on/off client toolbar
TMACK	C↔S	—	yes	JSON	Acknowledge a timer request
TMRQ	C↔S	TMACK/FAILED	no	JSON	Request a timer
TO	C↔S	—	no	JSON	Send chat message
UPDATES	C←S	—	N/A	JSON	Advertise software updates
WORLD	C←S	—	N/A	JSON	Campaign world info

Table F.2: Summary of Client/Server Messages (continued)

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.

The mapper client will interpret a server comment beginning with the text "notice:" which is received during the initial connection negotiation (i.e., before the READY message is given by the server) as a notice that the end user should see. The text following the word "notice:" is posted in an alert box. Clients MUST recognize the string in all lower-case if they choose to implement this feature; they MAY also recognize it regardless of case.

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. The reason for this is that this data format is valid in all protocol versions, including those before version 400, so it can be used to unambiguously identify the server's protocol to all possible clients.

  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 random ones. 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:

`Animation` *(object)*
: If this value is given and non-null, this image is to be animated from individual frames. If the client is capable of animating multi-frame files, the base filename can be used directly. Servers offering this file SHOULD provide the multi-frame animated image file in a variety of formats, and SHOULD provide single-frame files in all formats in which they provide static images.

  Individual frame files are named by prefixing the base filename with "$:\langle n \rangle:$" where $\langle n \rangle$ is the frame number, starting with 0. The `Animation` object has the following fields:

  `Frames` *(int)*
  : The number of frames in the animation.

  `FrameSpeed` *(int)*
  : The number of milliseconds to delay between frames.

  `Loops` *(int)*
  : If present and non-zero, the animation will play this many times before stopping. Otherwise, the animation plays forever.

`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` or ⟨*base*⟩`/a/ab/abcdefg.png` 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` (JPEG format) or `a/ab/abcdefg.png` (PNG format) may be retrieved for clients which use those other formats. Currently, `mapper.tcl` uses GIF or PNG 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.

`AI/`

Filters (removes) all the saved image definitions held in the server's database whose names match a regular expression. The payload is a JSON object with the following fields:

`Filter`
: The regular expression used to select which image names will be removed.

`KeepMatching` *(bool)*
: If true, only those images whose names match the `Filter` expression will be kept; the others will be removed. Otherwise, the opposite is done: images whose names match `Filter` will be removed.

`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, the following one features are 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.

    `DICE-COLOR-LABELS`
    : The client supports formatting controls in die-roll label fields in the roll details in a format as described above for subtitles. This allows custom colors to be used for special die-roll modifiers that need to stand out visually from all the rest.

    `GMA-MARKUP`
    : The client is capable of interpreting `GMA` markup formatting codes received in chat messages.

`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 authenti-

cation 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` field 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 sent by the server),
i is the number of hash rounds to apply,
$H(x)$ 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 the strings x and y,

then the procedure to calculate the `Response` string is:

1. If $i = 0$ or was not provided by the server, 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:
   a) Calculate $D' = H(P||D)$.
   b) 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 $-\langle target \rangle$ messages (e.g., a $\langle target \rangle$ 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 the form "[⟨*imagename*⟩=]⟨*name*⟩") may be used, which removes a creature token whose display name matches the `name` 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.

`CORE`
: Query the server's core SRD database. The server SHOULD respond with a `CORE=` message containing the requested item or indicating that no such item was found. The payload is a JSON object with the following attributes:

  `Type`
  : The type of object being requested. Currently the supported types include `bestiary`, `class`, `feat`, `language`, `skill`, `spell`, and `weapon`.

  `Code`
  : The GMA unique code that identifies the entry. May be blank since only one of `Code` and `Name` is necessary.

  `Name`
  : The common name of the entry. May be blank since only one of `Code` and `Name` is necessary.

  `RequestID`
  : If you place an arbitrary string value in this field, it will be sent back to you in the corresponding `CORE=` response message.

`CORE=`
: This message from the server provides core SRD data for an item that the client might want to know about. Typically, this is in response to a `CORE` message from the client requesting this information. Clients MUST gracefully deal with the receipt of a `CORE=` message they didn't need or are no longer interested in, as well as the case where they requested core data but received no corresponding response. The payload is a JSON object containing the requested data, with the following attributes:

  `NoSuchEntry` *(bool)*
  : If true, the requested entry was not found in the database. Otherwise (including the case where this attribute is missing) the remaining attributes contain the requested information.

  `IsHidden` *(bool)*
  : If true, the requested entry is hidden from player view by the GM. No `Data` field will be given.

`IsLocal` *(bool)*
: If present and true, this entry was a local addition to the database. Otherwise it was imported from publicly-available `SRD` data for the game system.

`Code`
: The `GMA` code for the requested item.

`Name`
: The `GMA` name for the requested item.

`Type`
: The type of core data item being provided, as per the `Type` field of the `CORE` message (q.v.).

`RequestID`
: If present and non-empty, this is the `RequestID` value provided by the client when making the request that resulted in this reply.

`Data`
: This value is a `JSON`-encoded object containing the specific data relating to the item in question. Its format depends on the item `Type`.

  The `Data` payload for each type is described in the following sections. **N.B. The descriptions that follow of the various Data objects are speculative at this point and entirely subject to change.**

`Data` fields for `bestiary` type items:

`ImageTag`
: If the `GM` maintains a library of creature token images for their local bestiary, and one is defined for this creature, then this field will contain the image tag name for this creature's token.

`Species`
: The name of the species of the creature.

`CR`
: The challenge rating of the creature. Fractions may appear as, e.g., `1/2`.

`XP` *(int)*
: `XP` reward for defeating the creature.

`Class`
: The class and levels if applicable.

`Alignment` *(object)*
: The alignment for the creature, as an object with the following fields:

  `Alignments` *(list)*
  : A list of alignment codes (`CG`, `LN`, etc.) which this creature may be.

`Special`
: Any special comment about alignment.

`Source`
: The reference to the source this item originally came from.

`Size` *(object)*
: The creature's size, as an object with the following fields:

  `Code`
  : The size code as documented in the GMA encounter and familiar data.

  `SpaceText`
  : Any comment about the creature's threatened space.

  `ReachText`
  : Any comment about the creature's reach distance.

`Type`
: Creature type.

`Subtypes` *(list)*
: List of applicable subtypes.

`Initiative` *(object)*
: The creature's initiative modifier, as an object with the following fields:

  `Mod` *(int)*
  : The total initiative modifier value.

  `Special`
  : Any special information related to initiative.

`Senses`
: A description of the creature's senses.

`Aura`
: A description of the creature's aura effect.

`HP` *(object)*
: The creature's hit points as an object with the following fields:

  `Typical` *(int)*
  : Typical hit point total.

  `Current` *(int)*
  : The creature's current hit point total.

  `Special`
  : Any special comments about the creature's health.

  `HitDice`
  : The hit dice die-roll specification.

`Save` *(object)*
: The creature's saving throws, as an object with these fields:

  `Fort` *(object)*
  : The Fortitude saving throw, as an object with these fields:

`Mod` *(int)*
The total saving throw modifier value.

`Special`
Any special commentary about this saving throw.

`NoSavingThrow` *(bool)*
If true, the creature has no such saving throw at all and the other fields should be ignored.

`Refl` *(object)*
The creature's Reflex saving throw, in the same format as the Fortitude saving throw.

`Will` *(object)*
The creature's Will saving throw, in the same format as the Fortitude saving throw.

`Special`
Any special comments about the creature's saving throws.

`DefensiveAbilities`
A description of the creature's defensive abilities.

`DR` *(object)*
The creature's damage reduction, as an object with the following fields:

`DR` *(int)*
The number of hit points resisted.

`Bypass`
Description of what will bypass the `DR`.

`Immunities`
A description of the things the creature is immune from.

`Resists`
A description of the resistances the creature has.

`Speed` *(object)*
The creature's movement speed, as an object with the following fields:

`Code`
The encoded list of speeds.

`Special`
Additional commentary about movement speed.

`SpecialAttacks`
A description of the various special attacks the creature may make.

`Resists`
A description of the resistances the creature has.

`Speed` *(object)*
The creature's movement speed, as an object with the following fields:

`Code`
: The encoded list of speeds.

`Special`
: Additional commentary about movement speed.

`SpecialAttacks`
: A description of the various special attacks the creature may make.

`Abilities` *(object)*
: The creature's ability scores, as an object with the following fields:

  `Str` *(object)*
  : The Strength score, as an object with the following fields:

    `Base` *(int)*
    : The raw ability score value.

    `Special`
    : Any special notes about this ability score.

    `NullScore` *(bool)*
    : If true, the creature does not have this ability score at all, and the other fields should be ignored.

  `Dex` *(object)*
  : The Dexterity score, as an object in the same format as Strength.

  `Con` *(object)*
  : The Constitution score, as an object in the same format as Strength.

  `Int` *(object)*
  : The Intelligence score, as an object in the same format as Strength.

  `Wis` *(object)*
  : The Wisdom score, as an object in the same format as Strength.

  `Cha` *(object)*
  : The Charisma score, as an object in the same format as Strength.

`Combat` *(object)*
: The combat statistics for the creature, as an object with the following fields:

  `BAB` *(int)*
  : The base attack bonus.

  `CMB` *(int)*
  : The combat maneuver bonus.

  `CMD` *(int)*
  : The combat maneuver defense value.

`CMBSpecial`
: Any special notes on the CMB.

`CMDSpecial`
: Any special notes on the CMD.

`SQ`
: A description of the creature's special qualities.

`Environment`
: A description of the environment where this creature commonly appears.

`Organization`
: A description of the groups in which this creature is usually found.

`Treasure`
: The treasure type and amount usually found with this creature.

`Appearance`
: A short description of the creature's physical appearance.

`IsTemplate` *(bool)*
: If true, this is a template from which other creatures can be built.

`Strategy` *(object)*
: The creature's strategy in an encounter, as an object with these fields:

  `BeforeCombat`
  : How the creature will prepare for the encounter.

  `DuringCombat`
  : How the creature will manage its encounter with the players.

  `Morale`
  : How resolute the creature is in the face of deadly combat.

`IsCharacter` *(bool)*
: If true, this is a creature that may be a character.

`IsCompanion` *(bool)*
: If true, this creature may be a companion.

`IsUnique` *(bool)*
: If true, there is only one of this creature in existence.

`AgeCagetory`
: The age category for the creature.

`Gender`
: The creature's gender

`Bloodline`
: The creature's bloodline.

`Patron`
: The creature's patron deity.

`AlternateNameForm`

`DontUseRacialHD` *(bool)*
: If true, don't use racial hit dice when creating a character with levels.

`VariantParent`

`Mythic` *(object)*

`IsMythic` *(bool)*

`MR` *(int)*

`MT` *(int)*

`OffenseNote`

`StatisticsNote`

`Gear` *(object)*

`Combat`

`Other`

`Schools` *(object)*

`Focused`

`Prohibited`

`Opposition`

`ClassArchetypes`

`BaseStatistics`

`RacialMods`

`Mystery`

`Notes`

Domains *(list)*

AC *(object)*

Components *(object)*

Adjustments *(object)*

AttackModes *(list of objects)*

Tier *(int)*

BaseWeaponID

Multiple *(int)*

Name

Attack

Damage

Critical *(object)*

CantCritical *(bool)*

Threat *(int)*

Multiplier *(int)*

Ranged *(object)*

IsRanged *(bool)*

Increment *(int)*

MaxIncrements *(int)*

IsReach *(bool)*

Special

```
    Mode
Languages (list of objects)
    Name
    IsMute (bool)
    Special
Feats (list of objects)
    Code
    Parameters
    IsBonus (bool)
Skills (list of objects)
    Code
    Modifier (int)
    Notes
Spells (list of objects)
    ClassName
    CL (int)
    Concentration (int)
    NoConcentration (bool)
    PlusDomain (int)
    Description
    Special
```

`Spells` *(list of objects)*

`Name`

`AlternateName`

`Frequency`

`Special`

`Slots` *(list of objects)*

`IsCast` *(bool)*

`IsDomain` *(bool)*

`MetaMagic` *(list)*

`Data` fields for `class` type items:

`Spells` *(object)*
Describes the spell-casting capability of the class, with the following fields:

`Type`
The magic type (arcane, etc.)

`Ability`
The name of the ability score relevant to spells.

`HasBonusSpells` *(bool)*
True if the class allows bonus spells (e.g., clerics' domain spells)

`IsSpontaneousCaster` *(bool)*
True if the class casts spells without needing to prepare them in advance.

`CastPerDay` *(list)*
A list of the number of spells which may be cast per day. Each element is an object which has the following values:

`ClassLevel` *(int)*
The class level for which this applies.

`SpellLevel` *(int)*
The level of spell for which this applies.

`IsProhibited` *(bool)*
If true, this level of spell is not possible at this class level.

`IsUnlimitedUse` *(bool)*
If true, this level of spell can be used an unlimited number of times per day at this class level.

`Number` *(int)*
The number of spells granted at this spell and class level.

`PreparedPerDay` *(object)*
The number of spells which may be prepared per day by class level and spell level. Each element has the same values as for the `CastPerDay` attribute above.

`SpellsKnown` *(object)*
The number of spells which may be known by class level and spell level. Each element has the same values as for the `CastPerDay` attribute above.

`Data` fields for `feat` type items:

`Parameters`
Describes the parameters allowed for the feat, if any.

`Description`
The text description of the feat.

`Prerequisites`
The text description of the prerequisites for the feat.

`Benefit`
Description of the benefit granted by taking this feat.

`Normal`
Description of the situation if the feat is not taken.

`Special`
Any special notes or circumstances.

`Source`
The reference to the source book or other place where this feat origially came from.

`Race`

`Note`

`Goal`

`CompletionBenefit`

`SuggestedTraits`

`Types` *(list)*
A list of feat types to which this belongs.

`FlagNames` *(list)*
: A list of flags which apply to this feat.

`MetaMagic` *(object)*
: If this is a metamagic feat, the following fields will appear in the `MetaMagic` attribute:

  `IsMetaMagicFeat` *(bool)*
  : If true, this is a metamagic feat and the other fields here will apply.

  `Adjective`

  `LevelCost` *(int)*
  : The number of levels this feat costs (i.e., the number of spell slots higher than normal this spell will use).

  `IsLevelCostVariable` *(bool)*
  : If true, the spell level cost is not fixed.

  `Symbol`
  : The symbol to use on GMA character record sheets and encounter run sheets for spells which were prepared with this feat.

`Data` fields for `language` type items:
: There is no additional data. The language name appears in the `Name` field.

`Data` fields for `skill` type items:

  `ClassSkillFor` *(list)*
  : A list of the class codes for which clases this skill is a class skill.

  `Ability`
  : Relevant ability score name.

  `HasArmorPenalty` *(bool)*
  : If true, this skill is subject to an armor check penalty.

  `TrainingRequired` *(bool)*
  : If true, the character must be trained to use this skill.

  `Source`
  : The reference to the original source for this skill.

  `Description`
  : The abbreviated text description of the skill.

  `FullText`
  : The full text description of the skill.

  `ParentSkill`
  : The name of the parent skill if applicable.

  `IsVirtual` *(bool)*
  : If true, this skill cannot be taken; one of its children must be taken specifically.

`IsBackground` *(bool)*
: If true, this is a background skill.

`Data` fields for `spell` type items:

`School`

`Descriptors` *(list)*

`Components` *(object)*

- `Components` *(list)*
- `Material`
- `Focus`
- `HasCostlyComponents` *(bool)*
- `MaterialCosts` *(int)*

`Casting` *(object)*

- `Time`
- `Special`

`Range` *(object)*

- `Range`
- `Distance` *(int)*
- `DistancePerLevel` *(int)*
- `DistanceSpecial`

`Effect` *(object)*

- `Area`
- `Effect`

    Targets

Duration *(object)*

    Duration

    Special

    Concentration *(bool)*

    PerLevel *(bool)*

SR *(object)*

    SR

    Special

    Object *(bool)*

    Harmless *(bool)*

Save *(object)*

    SavingThrow

    Effect

    Special

    Object *(bool)*

    Harmless *(bool)*

IsDismissible *(bool)*

IsDischarge *(bool)*

IsShapeable *(bool)*

ClassLevels *(list of objects)*

`Class`

`Level` *(int)*

`Deity`

`Domain`

`Description`

`Source`

`Bloodline`

`Patron`

`Data` fields for `weapon` type items:

`Cost` *(int)*
: The cost in copper pieces for a standard weapon of this type.

`Damage` *(object)*
: The damage done by this weapon. The value is an object where the keys are size category codes and the values are the damage done by a weapon of that size category.

`Critical` *(object)*
: The critical hit stats for the weapon. This is an object with the following fields.

  `CantCritical` *(bool)*
  : If true, this weapon cannot score a critical hit at all. The other fields should be ignored.

  `Multiplier` *(int)*
  : The number of extra dice to add when a critical hit is scored.

  `Threat` *(int)*
  : The minimum die value to threaten a critical hit (e.g., 19 means to threaten on a natural 19–20).

`Ranged` *(object)*
: For ranged weapons, the fields in this attribute describe those parameters.

  `Increment` *(int)*
  : The range increment for the weapon.

`MaxIncrements` *(int)*
The maximum number of increments before the target is out of range.

`IsRanged` *(bool)*
If true, this is a ranged weapon and the other fields are meaningful.

`Weight` *(int)*
The weight in grams of the weapon.

`DamageType` *(list)*
The type of damage done. This is a list of strings, with each indicating a damage type such as `B` for bludgeoning, `P` for piercing, and `S` for slashing.

`Qualities` *(list)*
A list of weapon qualities. The codes in the list may include: `1` (one-handed), `2` (two-handed), `b` (brace), `D` (disarm), `d` (double), `E` (Elven), `f` (fragile), `G` (Gnome), `g` (grapple), `H` (Halfling), `L` (light), `M` (monk), `m` (martial), `O` (Orc), `R` (ranged), `r` (reach), `S` (non-lethal), `s` (simple), `t` (trip), `U` (unarmed), `W` (Dwarven), `X` (exotic), `x` (deadly), and `Z` (masterwork).

`CORE/`
Filter the core database items which are visible to the players. The payload is a `JSON` object with the following fields:

`Filter`
This is a regular expression matching the `Code` value of all the entries to be affected by this command.

`Type`
The type of core data entry to target.

`IsHidden` *(bool)*
If true, set the entries matching the `Filter` expression so they are hidden from the players. Otherwise, make them visible.

`InvertSelection` *(bool)*
If true, all entries which do *not* match `Filter` are affected.

`COREIDX`
Query the core database to retrieve a list of entries. This gives the information by which the actual data behind these entries may be retrieved using the `CORE` message. The payload includes the following fields:

`Type`
The type of entry to index, as per the `CORE` message.

`CodeRegex`
: If present and non-empty, limit the responses to those whose `Code` field matches this regular expression.

`NameRegex`
: If present and non-empty, limit the responses to those whose `Name` field matches this regular expression.

`Since` *(RFC 3339-formatted datetime string)*
: If present and non-empty, limit the responses to those which were modified since the given date and time.

`RequestID`
: If a string value is provided here, it will be repeated in the server's response message.

`COREIDX=`
: The server response to a `COREIDX` message. Since the number of entries in the database may be quite large, the server will send a number of these reply messages, one per item from the database. It includes the following fields:

  `RequestID`
  : The `RequestID` sent to the server in the `COREIDX` request that initiated this update, if any.

  `Type`
  : The database item type

  `IsDone` *(bool)*
  : If true, this is the final response in this set.

  `N` *(int)*
  : The number of response this is, starting with 1.

  `Of` *(int)*
  : The total number of responses that will be given.

  `Name`
  : The item's name.

  `Code`
  : The item's code.

`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`

Define (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, except that it does not contain `DelegateFor` or `Delegates` fields.

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

`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 and all authorized delegates.

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 and all authorized delegates. The JSON payload includes the following fields:

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

Global *(bool)*
: If present and true, this operation filters items out of the system-wide global set of presets instead of a user's set. This makes this a privileged command.

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 and authorized delegates 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:

For
: The username of the player for whom these presets were stored. The default, if no name is given or if it is empty, is the currently logged-in user.

Global *(bool)*
: If present and true, these presets are the system-wide set that apply to all users rather than those of the target user. For the DD command, this means the operation is redefining the entire set of system-wide presets. For DD+, the operation is adding to the system-wide presets without changing the other existing ones.

  Note that when any of these commands are sent with this parameter set to true, they become privileged commands.

  In the DD= response from the server, the presence of this parameter with a true value indicates that this contains only the system-wide presets. In this case, clients MUST NOT conclude that the

absence of the user's personal presets in the payload means it can delete its copy of the user's presets. Without the `Global` parameter, this message contains all presets the client should know about, including the user's set plus the system-wide globals.

`DelegateFor` *(list of strings)*
: The list of all the other users for whom the target user is a delegate.

`Delegates` *(list of strings)*
: The list of delegates who also have visibility and control for the owner's presets. If empty or missing, there are no delegates.

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

  `Global` *(bool)*
  : If this parameter is present and true, it indicates that this preset was defined at a system-wide level and may not be changed or deleted by the user. The client MUST track it separately and MUST NOT confuse it with a non-global preset with the same name. This parameter is only set by the server. Clients MAY NOT set this when defining or modifying presets and sending them to the server.
    We RECOMMEND clients use a notation to make system-wide and per-user preset names distinct to the user. For example, the GMA Mapper uses single prefix characters on names for variables and tables as in `$foo` and `#bar`, but double prefix characters for system-wide names, as in `$$foo` and `##bar`.

  `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. The formatting of the `Name` field is not mandated by this specification. However, the following standard interpretation for data to the left of the vertical bar (if any) is RECOMMENDED for clients using this feature:

    1. Regular die-roll presets with a preferred display ordering SHOULD have a three-digit decimal zero-padded sequence number to the left of the bar, e.g., "`000|attack`", "`001|damage`", "`002|fort save`", etc. The numbers need not be strictly consecutive. Clients SHOULD use dictionary sorting for preset items so that the numeric sort-order sequence value is correctly arranged regardless of number of characters; however the recommendation here

is also to pad them with zeroes to three digits for the benefit of clients which cannot or will not do so. Clients MAY include optional ⟨*flags*⟩ and ⟨*client_data*⟩ fields delimited semicolons following the sequence value (e.g., "`042;b|example`"). This contains a number of single-character flags. The following flag is currently recognized:

`b`
: Send the die roll blindly to the GM only.
This means the full definition of the simple preset name is:
⟨*sequence*⟩`[;`⟨*flags*⟩`[;`⟨*client_data*⟩`]]|`⟨*name*⟩

2. Stored variables MAY be stored as a die-roll preset with a `Name` field in the format
"`§`⟨*sequence*⟩`;`⟨*var*⟩`;`⟨*flags*⟩`[;`⟨*client*⟩`]|`⟨*displayname*⟩".
where `§` is Unicode character U+00A7 (section symbol), ⟨*sequence*⟩ is a three-digit decimal zero-padded display sequence number as described above, ⟨*var*⟩ is the name of the variable by which the contents of this preset may be inserted into another die-roll specification, ⟨*flags*⟩ are a set of zero or more single-letter flags affecting how the client should use this value, and ⟨*client*⟩ is an optional field with client-specific data not otherwise defined here. If the ⟨*client*⟩ field is not given, the semicolon immediately before it may be omitted as well. For example:
"`§042;CL;e|caster level`"
The ⟨*flags*⟩ are up to the client to interpret, but we RECOMMEND the flag "`e`" to mean the value should be enabled (or "on") by default and "`g`" to mean the die-roll expression to the left of it should be grouped together in parentheses.
If the ⟨*var*⟩ value is empty (e.g., "`§000;;e|inspiration`"), clients SHOULD interpret this to mean that the preset's value should not be assigned to a variable name, but should be appended to any die-roll specification made while it is enabled.

3. It is RECOMMENDED that clients allow for grouping of related presets and/or modifiers by appending one or more group names to the end of the numeric sort order digits. Each such name is separated from the digits and each other with a right-pointing triangle "`▶`" (Unicode U+25B6). `012|attack` is an ungrouped attack roll, `013▶monster|attack` is an attack roll belonging to the "monster" group, `030▶monster▶save|Fortitude` is a Fortitude saving throw belonging to the "save" subgroup of the "monster" group, and `§042▶monster;;|mod` is a modifier in the

"monster" group. Clients SHOULD allow users to collapse all of the die rolls with the same group name(s) appearing consecutively in the die-roll preset list.

4. Stored tables MAY be stored as a die-roll preset with a `Name` field in the format
   `#`⟨*sequence*⟩`;`⟨*flags*⟩`[;`⟨*client*⟩`]|`⟨*tblname*⟩
   and the `DieRollSpec` field in the format
   `;`⟨*dieexpr*⟩`;`⟨*n0*⟩`;`⟨*t0*⟩`;...;`⟨*nN-1*⟩`;`⟨*tN-1*⟩`;*;`⟨*tN*⟩`;`
   The field delimeter ("`;`" in this example) is arbitrary but must be consistent; it is whatever character appears as the first character in `DieRollSpec` (to allow for punctuation marks such as semicolons to appear in the table text as needed). The semicolon delimeters in the `Name` field are always semicolons. This declares a lookup table named ⟨*tblname*⟩ (which MUST be unique within the preset list). The set of ⟨*flags*⟩ is up to the client to manage. Currently the defined flags are:

   `b`
   : The die rolls should be send blindly to the GM only.

   `m`
   : the table's text may include GMA markup formatting codes

   The ⟨*dieexpr*⟩ gives a standard die-roll expression which the client should send to the server whenever a roll on this table is requested. Following this are a set of cases represented by numeric values ⟨*n*⟩ in increasing order along with corresponding text strings ⟨*t*⟩ which may contain GMA markup formatting codes if the `m` flag is present. The last case, which is taken if the result of the die roll exceeded any of the values of ⟨*n*⟩ found up to that point, is represented by "*".
   When the user wishes to invoke a random roll using the lookup table, the client will send a die-roll request to the server with ⟨*dieexpr*⟩ (along with, perhaps, additional modifiers and options) to get a random result back, which is then compared to the values in the lookup table until a value is found that is greater than or equal to the result, or the final "*" value is reached, whereupon the corresponding text string is displayed by the client to the user in some appropriate fashion. Clients SHOULD transmit this result to other clients via a TO message.

5. Clients MAY encode arbitrary dictionaries of complex data into presets by encoding the `Name` field as
   `©[`⟨*sequence*⟩`[;`⟨*client_data*⟩`]]|`⟨*name*⟩
   where ⟨*sequence*⟩ is an optional sequence and/or group

hierarchy as described above, and ⟨*client_data*⟩ is an arbitrary string to be interpreted by other programs consuming this preset data as they choose. In this case, the `DieRollSpec` field MUST be a properly-formed JSON object string with no newline characters that encodes the data being stored in this preset. The "©" character is the Unicode codepoint U+00A9.

6. Clients MUST gracefully handle the existence of data formats not listed above without losing or corrupting any presets or metadata.
   Notably, the `gma`(6) program itself automates the creation and destruction of ephemeral presets for combatants on an encounter-by-encounter basis. The names for these presets include an additional component, ⟨*area*⟩, which identifies the dungeon location or encounter identifier. To accomplish this, GMA prepends the string `$[`⟨*area*⟩`]` to the ⟨*client_data*⟩ field for any of the complex types which support that. Otherwise it prepends that string to the ⟨*sequence*⟩ value.

`Description`
: A text description of the purpose of the preset.

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

`DDD`
: Define Die-roll Delegate. Clients send this command to the server to indicate that other user(s) are allowed to retrieve and store their die-roll presets. The server SHOULD immediately send `DD=` commands to the requesting user and all delegates. From this point on, any updates to the requesting user's presets will result in `DD=` commands sent to all delegates as well. The JSON payload includes:

  `For`
  : The user for whom these delegates are being defined. If empty or missing, the default is to define them for the requesting user. Only the GM may set delegates for someone else.

  `Delegates` *(list of strings)*
  : The list of usernames who are allowed to access the requesting user's die-roll presets. This replaces the previous set of delegates. If the list is empty (or the `Delegates` field missing), then all delegates are removed.

`DENIED`
: Access to the server is denied. The server will send this to a client; clients MAY NOT send it. When a client receives this message, it MUST

not continue to communicate with the server. In fact, the server SHOULD terminate the network connection after sending the DENIED message. 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. The JSON payload may contain the following field:

`Global` *(bool)*
: If present and true, this requests a list of only the global, system-wide set of presets rather than all of the presets the client should know about.

`For`
: The name of the user whose presets are being requested. Only the GM or an authorized delegate may request other users' presets. If this field is empty or missing, the requesting user's presets are requested.

`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 damage`, `ability drained`, `bleed`, `blinded`, `confused`, `cowering`, `dazed`, `dazzled`, `dead`, `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 symbols shown in Table F.3.

Symbol	Description of Indicator
`|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".)

Table F.3: Condition Marker Symbols

`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.

`Transparent` *(bool)*
: If true, the creature token should be displayed in a semi-transparent form whenever it has this condition (added in protocol 404).

`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.

    The server will also populate the following additional fields in its reply, which a client can use to calculate the round-trip time for the request and identify how much of that was internal to the server and how much was network latency.

  `ReceivedTime` *(RFC 3339-formatted datetime string)*
  : The time when the server received the packet from the client.

  `SentTime` *(RFC 3339-formatted datetime string)*
  : The time when the server sent the response packet to the client.

`FAILED`
: Gives notice that a client request was not accepted. This may be issued by the server for reasons such as an error in the request itself that the server is able to detect before processing it further, or it may

originate from a GM client in response to a player request such as TMRQ where the request contains an error or the GM decides to decline the request. If this message is sent by a client, it MUST be a privileged client, and will be relayed to the indicated requesting client. The JSON payload includes the following fields:

`IsError` *(bool)*
: If true, this notice indicates that the request failed because it contains an error that prevents it from being carried out.

`IsDiscretionary` *(bool)*
: If true, this notice is the result of a decision by the GM to decline the player's request.

`Command`
: The original command being referenced.

`Reason`
: Freeform text describing why the request failed or was declined. This should be all that needs to be shown to the end user to explain the issue.

`RequestID`
: The uniquie identifier sent along with the original request.

`RequestedBy`
: The user name of the original requester.

`RequestingClient`
: The client ID of the original requester. When the FAILED message is sent by a GM client, the server will relay it to this client session.

`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.

`HPACK`
: Acknowledge a hit point change request made via a previous HPREQ message. The JSON payload for this message has the following fields:

  `RequestID`
  : The ID of the original request, as sent by the client when making it.

  `RequestingClient`
  : The client connection ID as it appeared in the original request.

`RequestedBy`
: The user name of the original requester. If the server is unable to locate the original requesting client connection as specified by the `RequestingClient` field, it MAY send `HPACK` messages to clients logged in by the user named in this field instead.

`HPREQ`
: Request an update to a target creature's hit points. This request will be sent to the GM for approval. If the GM approves, the creature's hit points will be amended in the system and will be reflected as appropriate in the tracking information sent by the server. The client SHOULD also receive a corresponding `HPACK` message acknowledging this request. Clients MUST cope with the possibility of not receiving those acknowledgement messages or there being long delays between the request and acknowledgement. If the GM declines to accept the request, or if the request contains an error, a `FAILED` message SHOULD be sent back to the requesting client with the same `RequestID` value as the original request. The JSON payload for this message includes the following fields:

  `Targets` *(list of strings)*
  : The names of the creatures as they appear on the map. This should hold a single value when requesting an update to a character's hit point total, but may, for example, be useful to specify multiple characters at once for communal spells granting many people at once a block of temporary hit points.

  `Description`
  : Text description of what this request is for, where it came from, etc.

  `RequestedBy`
  : The user name of the user who initiated this request. Clients NEED NOT supply this field; it will be supplied by the server as it relays the request on to the GM.

  `RequestingClient`
  : The client connection ID of the user who initiated this request. Clients MUST NOT supply this field; it will be supplied by the server as it relays the request on to the GM. The meaning of this value is opaque and is only used by the server.

  `RequestID`
  : A unique identifier for this request. Replies sent back to the requesting client in regard to this request will reference this same ID. A UUID string is suggested for this value.

  `Healh` *(object)*
  : The health object values you wish to set. If this is `null`, no changes are requested. Otherwise, the only values which will be considered for this request are:

`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.

`TmpHP` *(object)*
: A request to add temporary hit points to the creature. If `null`, no temporary hit points are being requested. Otherwise, the following fields are expected:

  `TmpHP` *(int)*
  : The number of temporary hit points to add to the creature's temporary hit point pool.

  `TmpDamage` *(int)*
  : The number of temporary hit points already removed due to damage (usually this will be zero, but just in case this request needs to account for something that's already happened, this may be noted here).

  `Expires` *(duration)*
  : The time remaining until the temporary hit points expire, in the same format as for the `Expires` field in the `TMREQ` message (q.v.).

`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 the 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 tothe `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.

`Stipple`
: If nonempty, the value is the name of a bitmap image which is assumed to be known to the client. The shape will be filled in with the `Fill` color using this bitmap to form a stipple pattern in that color. If the `Stipple` bitmap is empty or unknown to the client, a solid color fill is used instead. If the `Fill` field is empty, then `Stipple` is ignored. Clients SHOULD by default understand stipple bitmap names "`gray12`", "`gray25`", "`gray50`", and "`gray75`".

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

`Layer`
: The map layer this element belongs to. Currently not implemented by the mapper client.

`Level`
: The dungeon level where this element appears. Currently not implemented by the mapper client.

`Group`
: The object group to which this element belongs. Currently not implemented by the mapper client.

`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. Currently not implemented by the mapper client.

`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. The client MAY truncate this to an integer if it can't handle fractional point sizes.

    `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.

  `Iterations` *(int)*
  : The number of rounds of hashing to apply to generate the challenge response. This value SHOULD be in the range [64,4095].

  `ServerStarted` *(RFC 3339-formatted datetime string)*
  : The absolute wall-clock time that the server was started.

  `ServerActive` *(RFC 3339-formatted datetime string)*
  : The abolute wall-clock time of the server's last `MARCO` ping signal. If this was long ago (more than a minute or two with out-of-the-box server configuration), it is an indicator that the server's main event loop is deadlocked.

  `ServerTime` *(RFC 3339-formatted datetime string)*
  : The current wall-clock time on the server. This provides an accurate point of reference to determine how long ago the other server time values occurred.

  `ServerVersion`
  : The server's version number. (Added in protocol 405.)

`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 fields:

  `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.

  `Targets` *(list)*
  : If this progress bar is associated with a specific character or characters, they are listed by name here.

  `IsTimer` *(bool)*
  : If true, this is showing the status of a running timer instead of an operation in progress.

`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. The value may fluctuate up or down over the course of the updates that are received. It may also go negative. It is up to the client to display negative values appropriately based on the type of progress meter or other context. For example, timers generated by GMA will run negative for a number of updates to show that the timer has expired before sending a final update with `IsDone` true to indicate that the timer should be removed from view.

`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.

As a special case, the server may send this message with `OperationID=*` and `IsDone=true`. This is a signal to clients that all timers should be cancelled and dismissed from view. If a subsequent `PROGRESS` message is sent with the `OperationID` of a timer that was dismissed in this way, clients should treat it like a new timer being created at that point.

`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.

`TmpHP` *(int)*

The creature's temporary hit points, in addition to their maximum.

`TmpDamage` *(int)*

The number of hit points worth of damage suffered against the temporary hit point total.

`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.

`Hidden` *(bool)*
: If true, this creature token is hidden from player view, although the `GM` can still see it. Added in protocol 404.

`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.

`PolyGM` *(bool)*
: If true, only the `GM` may access the polymorph capabilities of this creature. Map clients not logged in as `GM` `MUST` not display any information indicating that there are different skins available, nor allow the user to select among them.

`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 a value like `["M:base","M:disguise","L:giant"]`.

`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. This has the general form

  ⟨*category*⟩[⟨*natural*⟩][`->`⟨*extended*⟩][`=`⟨*space*⟩][`:`[`*`]⟨*comment*⟩]

  where ⟨*category*⟩ is a single-letter category code, ⟨*natural*⟩ is a custom natural reach distance in feet (if different than the standard for that size category), ⟨*extended*⟩ is a custom extended reach distance in feet (if different than the standard for that size category), and ⟨*space*⟩ is a custom creature space diameter in feet (if different than the standard for that size category). (Changed in protocol 406.)

  This is now DEPRECATED. If present and the `SkinSize` field is also present, then the value of `Size` MUST equal the value of the first element of `SkinSize`. The preferred usage going forward is to not set the `Size` field at all, and just use `SkinSize` alone. If there is only one skin for the creature, this value will have only a single value.

`DispSize`
: If non-empty, this indicates that the creature is temporarily resized to the given size category and should be displayed as such. The value is otherwise identical to `Size`. (Added in 406.)

`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.

`CustomReach` *(object)*
: Specify a custom reach distance for this creature, if different than the standard for a creature of its size category. This contains the following fields:

  `Enabled` *(bool)*
  : If true, there is a custom reach distance as described in the other fields. if false, the other fields should be ignored.

  `Natural` *(int)*
  : The number of feet from the perimeter of the creature token to which the creature's natural reach extends.

  `Extended` *(int)*
  : As `Natural` but indicates the distance for reach weapons.

  (Added in protocol 406.)

`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.

`REDIRECT`
: Instructs the client to use a different server for this session. This is used when the GM wants to use an alternate server temporarily without requiring the players to reconfigure all their clients. The server SHOULD send this during the initial stage of client interaction to avoid authenticating to a server you won't be using and then authenticating to the other one, but MAY be sent at any point. The client SHOULD disconnect immediately upon receiving this command, so the rest of the server negotiation may not even happen at all. The JSON payload includes the following values:

`Host`
: The host name or IP address of the server to connect to for this session.

`Port` *(int)*
: The TCP port number on which to connect to the server.

`Reason`
: The reason for the redirection to a new server.

This command SHOULD appear before any commands which would alter the client's internal state such as `DSM` or `AC`, because we cannot guarantee that the client will undo those settings before connecting to the new server which will send its own set of those commands. Ideally, when redirecting clients to a different server, `REDIRECT` should be the first item in the server's init file.

`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.

`Origin` *(bool)*
: If present and true upon receipt of this message from the server, this indicates that the client receiving it was the same client who originated the die roll reqeust whose result is being reported here, as opposed to a client who is merely being given a copy of the results of another client's die roll.

`Replay` *(bool)*
: If present and true upon receipt of this message from the server, this indicates that this message is being "replayed" (e.g., as a result of a `SYNC-CHAT` request from the client) and this is not the original sending of the message. As such, the client SHOULD NOT make active responses to it such as performing lookup table reports on it.

`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.

`Sent` *(RFC 3339-formatted datetime string)*
: The date and time the message was sent. Clients SHOULD NOT set a nonempty value for this field. It will be filled in by the server when sending messages to clients. If this field is missing or empty when received from the server, it means the date was not available to report for that message. Note that the special value "`0001-01-01T00:00:00Z`" is the "zero" value for dates and indicates that the date is not known (as is the case if this field is omitted entirely or is the empty string).

`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.

`TMACK`
: Acknowledges a timer request. If a client receives this, it means the GM has accepted their timer request into the set of tracked timers, and subsequent `PROGRESS` messages MAY appear as that timer's value is updated over time. The JSON payload for this message includes the following fields:

`RequestID`
: The ID of the original request, as sent by the client when making it.

`RequestingClient`
: The client connection ID as it appeared in the original request.

`RequestedBy`
: The user name of the original requester. If the server is unable to locate the original requesting client connection as specified by the `RequestingClient` field, it MAY send `TMACK` messages to clients logged in by the user named in this field instead.

`TMRQ`
: Request a new timer. This sends a request the the GM's GMA client. If the GM approves, the timer will be added to the collection of timers being tracked for the game. This MAY result in subsequent `PROGRESS` messages being received by clients as the timer's value is updated over time. If no GMA process hears the message, nothing may happen. Clients which receive this message MUST ignore it. If the GM declines to accept the incoming timer request, or if the request contains an error, a `FAILED` message SHOULD be sent back to the requesting client with the same `RequestID` value as the original request. If the GM accepts the timer into the tracking system, their client SHOULD send a `TMACK` message with matching `RequestID` field to the original requester to notify them of that fact. Clients MUST cope with the possibility of not receiving those acknowledgement messages or there being long delays between the request and acknowledgement.

  The JSON payload for this message includes the following fields:

`Description`
: A one-line text description giving the purpose of the timer.

`Expires`
: The time at which the timer should expire. This may be given as an absolute or relative time. If the string begins with an `@` character, it will be an abolute timer and the field MUST have the form

  @[[[⟨*year*⟩-]⟨*month*⟩-]⟨*date*⟩] ⟨*hour*⟩:⟨*minute*⟩[:⟨*second*⟩[.⟨*tenths*⟩]]

  where the ⟨*year*⟩, ⟨*month*⟩, ⟨*date*⟩, ⟨*hour*⟩, ⟨*minute*⟩, and ⟨*second*⟩ values are numeric and conform to the campaign world calendar currently in play, and ⟨*tenths*⟩ is a single digit representing tenths of seconds. For example, "`@15:32`" represents a timer which expires at 32 minutes past 3 P.M., and "`@10-15 12:00:05`" expires at five seconds past noon on the fifteenth day of the tenth month of the current year. Text names or abbreviations may be used for ⟨*name*⟩ if defined for the calendar in effect at the time, so on the Golarion calendar that last example could also have been "`@LAM-15 12:00:05`" since the tenth month is Lamashan.

Otherwise, the value indicates a relative amount of time from the time and date on the clock at the moment the timer is accepted into the system by the GM. This expiration MAY begin with a + or - sign. Following this is a number and a unit name, such as "`5 rounds`" or "`3 hours`", or an expression in the form

[[⟨*days*⟩:]⟨*hours*⟩:]⟨*minutes*⟩:⟨*seconds*⟩[.⟨*tenths*⟩]

so "`3 hours`", "`+3 hours`", "`3:00`", and "`+3:00`" all set a timer for three hours into the future, while "`-2 rounds`" sets one that expires two rounds in the past (which may not be as useless as it sounds depending on the circumstances in the game, but probably is).

`Targets` *(list of strings)*
: If this is empty, the timer is assumed to apply to everyone. Otherwise, it applies to the individuals named in the list. If these are the names of players, they should match their login names exactly.

`ShowToAll` *(bool)*
: If true, the timer should be visible to the players. Otherwise, only the GM sees it on their time tracker.

`IsRunning` *(bool)*
: If true, the timer should start running immediately. Otherwise, the GM will need to manually start it at a future point in time.

`RequestedBy`
: The user name of the user who initiated this request. Clients NEED NOT supply this field; it will be supplied by the server as it relays the request on to the GM.

`RequestingClient`
: The client connection ID of the user who initiated this request. Clients MUST NOT supply this field; it will be supplied by the server as it relays the request on to the GM. The meaning of this value is opaque and is only used by the server.

`RequestID`
: A unique identifier for this request. Replies sent back to the requesting client in regard to this request will reference this same ID. A UUID string is suggested for this value.

`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.

`Origin` *(bool)*
: If present and true upon receipt from the server, this means that the client receiving this message was the same client who sent the original message, instead of merely receiving a copy of the message sent by a peer client. Clients MUST NOT set this field.

`Replay` *(bool)*
: If present and true upon receipt of this message from the server, this indicates that this message is being "replayed" (e.g., as a result of a `SYNC-CHAT` request from the client) and this is not the original sending of the message.

`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.

`Sent` *(RFC 3339-formatted datetime string)*
: The date and time the message was sent. Clients SHOULD NOT set a nonempty value for this field. It will be filled in by the server when sending messages to clients. If this field is missing or empty when received from the server, it means the date was not available to report for that message. Note that the special value "`0001-01-01T00:00:00Z`" is the "zero" value for dates and indicates that the date is not known (as is the case if this field is omitted entirely or is the empty string).

`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`.

`Markup` *(bool)*
: If true, the text of this message may contain GMA markup formatting codes which the client SHOULD either filter out or interpret appropriately for display to the user. If a client did not indicate to the server that it was capable of understanding these codes via the `ALLOW` command (q.v.), then the server will filter out these codes before sending the message to that client.

`Text`
: The text of the message to be sent.

`Pin` *(bool)*
: If present and true, this message should be pinned in the recipients' chat windows as a persistent message until dismissed by the recipient.

`/CONN`
: 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", "gogma", or "core".

    `MinimumVersion`
    : If a server wishes to limit clients from this package to only those with a minimum version number (as self-reported by the client in its `AUTH` message), then a `MinimumVersion` and `VersionPattern` field MUST be added to that package's information here. The `MinimumVersion` field is a string with the minimum client version allowed to be used, as a semantic version string such as "`1.2`", "`1.7.3`", "`1.6-alpha.1`", etc. This will be matched against the value captured from the client's version number via the `VersionPattern` field. *This field should be used only in server init files. Clients SHOULD NOT use this field, nor expect it to be sent by the server.*

    `VersionPattern`
    : This gives a regular expression which is matched against the `Client` field sent by the client as part of its `AUTH` message when signing on to the server. This expression MUST contain a single capturing group which yields the client's version number to be compared against the `MinimumVersion` field described above. *This field should be used only in server init files. Clients SHOULD NOT use this field, nor expect it to be sent by the server.*

    `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. The version string MAY omit the third version number (e.g., “`4.2`” instead of the more standards-compliant “`4.2.0`”).

`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.

`ClientSettings` *(object)*
: Any of the following settings will override the client’s own settings. Clients SHOULD notify users that this has happened. This allows the GM to make global changes to things like where images are stored without requiring players to adjust all their settings.

  `MkdirPath`
  : The path to the `mkdir` program on the server (used for GM uploads of mapper content to the server).

  `ImageBaseURL`
  : The base URL from which the client will retrieve map and image files.

`ModuleCode`
: The current module's `ID` code.

`SCPDestination`
: The directory where `GM` uploads of mapper content should be sent to.

`ServerHostname`
: The hostname (and optionally username in the form *name*`@`*host*) for the `GM` to upload mapper content to the server.

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

SEE ALSO

`mapper`(5), `gma`(6), and `server`(6).

AUTHORS

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

HISTORY

This document defines map protocol version 419, which adds support for pinned chat messages.

Version 418 added support for temporary hit point tracking and user requests for hit point changes.

Version 417 introduced support for system-wide die-roll presets, table definitions and markup formatting in chat messages. It also added the `Origin` and `Replay` fields to the `ROLL` and `TO` messages.

Version 416 added the `FAILED`, `TMRQ`, and `TMACK` messages.

Version 415 added the `Iterations` parameter to the `OK` message.

Version 414 changed the definition of the `DENIED` message to allow it to be sent at any time during the conversation.

Protocol 413 added speculative definitions for the `CORE`, `CORE/`, and `CORE=` commands, and extensions to the `PROGRESS` command.

Protocol 412 added support for sharing die-roll presets.

Protocol 411 added timestamps to chat messages and die-roll results.

Protocol 410 added the `REDIRECT` command and the `ClientSettings` field of the `WORLD` command.

Protocol version 409 added the `Stipple` pattern fill attribute.

Protocol version 408 introduced the `PolyGM` creature attribute.

Protocol version 407 added the image animation features.

Protocol version 406 dropped the `Area` attribute from creatures and added the `CustomReach` attribute and the expanded syntax for the `Size` attribute, as well as the new `DispSize` attribute.

Protocol version 405 introduced the `ServerVersion` field to the server's greeting.

Protocol 404 introduced the `Hidden` attribute for creature tokens, the transparent image type, and the `Transparent` attribute for status markers.

Protocol version 403 introduced the `AI/` server message and additional fields to the `OK` message.

Protocol version 402 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.

gma-markup(7)

NAME

gma-markup – Simple text markup language used by GMA

DESCRIPTION

Various free-form text content within GMA, including spell descriptions, character sheet notes, and numerous others, allow for a simple markup language similar to Creole and Markdown (but distinct from either) to allow for casual formatting to be applied with a minimum of effort.

Lines are collected together into a single line which is wrapped as appropriate to the output format. A blank input line marks the division between paragraphs.

Basic Markup

`\\`
: Line break.

`\e`
: Literal "\" character.

`\v`
: Literal "|" character.

`//`⟨*text*⟩`//`
: Set ⟨*text*⟩ in italic type. [1,2]

`**`⟨*text*⟩`**`
: Set ⟨*text*⟩ in bold type. [1,2]

`\.`
: Zero-width space (does nothing, but it serves to disambiguate other items so they are not mistaken for markup syntax).

`<<->>`
: Page break (strictly speaking not part of the markup syntax, but some GMA text environments will break pages when seeing this symbol in the text).

Lists

`@`⟨*text...*⟩
: Starts bullet list item. [3]

`@@`⟨*text...*⟩
: Starts bullet sub-list item. [3]

`@@@`⟨*text...*⟩
: Starts bullet sub-sub-list item (et seq). [3]

`#`⟨*text...*⟩
: Starts numbered list item. [3]

`##`⟨*text...*⟩
: Starts numbered sub-list item. [3]

`###`⟨*text...*⟩
: Starts numbered sub-sub-list item (et seq). [3]

Links

`[[`⟨*item*⟩`]]`
: Create hyperlink to the named ⟨*item*⟩ in the GMA library (spells, feats, etc.).

`[[`⟨*item*⟩`|`⟨*name*⟩`]]`
: Create hyperlink to the named ⟨*item*⟩ but with link label ⟨*name*⟩.

Special Characters

Many common Unicode characters are recognized on input, but support for them in the target output format is not guaranteed. Additionally, the following markup symbols can be used to specify special characters.

`[S]`
: Section symbol: §

`[c]`
: Copyright: ©

`[R]`
: Registered: ®

`[<<]`
: Angle quote: «

`[>>]`
: Angle quote: »

`^o`
: Degrees: °

`+-`
: Plus/minus: ±

`^.`
: Bullet: •

`[0]`–`[9]`
: Superscript 0–9: ^{1}, ^{2}, etc.

`AE`
: AE ligature: Æ.

`ae`
: ae ligature: æ.

`1/4`
: Fraction 1/4: $\frac{1}{4}$.

`1/2`
: Fraction 1/2: $\frac{1}{2}$.

`3/4`
: Fraction 3/4: $\frac{3}{4}$.

`[x]`
: Multiplication sign: ×.

`[/]`
: Division sign: ÷.

`-`
: Hyphen: - (or minus sign: –).

`--`
: En dash: –.

`---`
: Em dash: —.

`‘`
: Open single quote: ‘.

`’`
: Close single quote: ’.

`‘‘`
: Open double quote: “.

`’’`
: Close double quote: ”.

`[+]`
: Dagger: †.

`[++]`
: Double dagger: ‡.

The letter `x` immediately next to a digit causes it to be printed as a multiplication sign (e.g., ×2 or 3×).

A hyphen immediately before a digit causes it to be printed as a minus sign (e.g., –1).

Separate numbers from fractions with an underscore (e.g., "`12_1/2`" prints as "$12\frac{1}{2}$").

Titles

`==[`⟨*title*⟩`]==`
: Set ⟨*title*⟩ as a major section title.

`==(`⟨*title*⟩`)==`
: Set ⟨*title*⟩ as a sub-section title.

Tables

Tables are specified by a set of lines which each begin with a `|` character in the far left position. Each column in the table is separated from the others with more `|` characters. A final `|` at the end of the row is optional.

`|=`⟨*text*⟩`|`
: Place ⟨*text*⟩ in a header cell (usually at the top of the table). [3]

`|`⟨*text*⟩`|`
: Place ⟨*text*⟩ in a regular table cell. If there is space between the left bar and the start of ⟨*text*⟩, then the text will be right-justified. If there is space before the bar to the right, the text will be left-justified. If there is space on both sides, it will be centered. [3]

`|- |`
: A cell with a `-` as the first character indicates that the cell to the left spans into this one. No contents of the cell are accepted after the hyphen. [3]

`|:`⟨*caption*⟩`|`
: Place ⟨*caption*⟩ on the table (may be above or below the table, depending on the output format). If multiple such rows are given, their contents will be combined into a single caption. Regardless of the number of columns for the table, the caption is typed as a single cell by itself on the line. [3,4]

`|::`⟨*footer*⟩`|`
: Adds a footnote to the table. If multiple such lines are given, they each add another footnote to the table. Regardless of the number of columns for the table, the caption is typed as a single cell by itself on the line. [3,4]

Inner Formatting

The text inside section headings and table cells may also contain a minimal kind of formatting. the `\\`, `**`, `//`, and `[[]]` formatting controls may be used inside these contexts, but due to limitations in some output media, the entire element may be set with a single font or other effect.

Notes

[1] May cross line boundaries but not cross paragraphs.

[2] May nest as in `//italic **and** bold//`.

[3] Must begin in the first column of the markup text.

[4] New as of version 7 (then backported to version 6).

SEE ALSO

`gma-markup`(6), `form-preamble`(7), `gma-preamble`(7).

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

The markup syntax was enhanced as of version 7.0.

Previously, `*` was used for both bullet list items and boldface, which was ambiguous. In version 7 this was resolved by making `@` the bullet list character, which has the side effect that a single `*` used at the start of a line, while not typeset in any special way, at least resembles a bullet list marker, so we have a “fail gracefully” mode of backward compatibility.

Version 7 introduced table captions and footers, as well as better handling of Unicode characters.

BUGS

Some of the choices for markup symbols are questionable. Others are arbitrary but based on some kind of history. For example, `\e` is the troff code for producing a backslash and `\\` is the LaTeX code for making a line break. The various ligatures such as `--` and `---` are also LaTeX-inspired.

gma-preamble(7)

NAME

gma-preamble.ps – GMA-Specific PostScript Form Preamble

DESCRIPTION

This is a PostScript preamble which sets up an easy API for printing GMA paper forms. It assumes `form-preamble`(7) has already been sent to the output device, and builds on the definitions of that file.

It sets a custom font encoding which allows us to use all the characters known to `gma-markup`(7).

PostScript Operator Notation

The documentation that follows uses the standard PostScript conventions for illustrating operators. The list of input values which are to be pushed onto the stack are listed first (or `--` if no input parameters are consumed by the operator), then the operator name itself, then the list of output values which the operator will push on the stack (or `--` if the operator doesn't push any values).

Fonts

`-- BISO` ⟨*font*⟩
: The Bold font with our special encoding. Defaults to Palatino-Bold.

`-- CISO` ⟨*font*⟩
: The fixed-width font with our special encoding. Defaults to Courier-Bold.

`-- HISO` ⟨*font*⟩
: The sans-serif font with our special encoding. Defaults to Helvetica.

`-- IBSO` ⟨*font*⟩
: The bold italic font with our special encoding. Defaults to Palatino-BoldItalic.

`-- IISO` ⟨*font*⟩
: The italic font with our special encoding. Defaults to Palatino-Italic.

`-- RISO` ⟨*font*⟩
: The Roman font with our special encoding. Defaults to Palatino-Roman.

`-- ZISO` ⟨*font*⟩
: The fancy font with our special encoding. Defaults to ZaphChancery-MediumItalic.

`-- DataFont` ⟨*font*⟩
The font to use for data values in the form. Defaults to `RISO`.

`-- DataFontBold` ⟨*font*⟩
The font to use for bold data values in the form. Defaults to `BISO`.

`-- DataFontBoldItalic` ⟨*font*⟩
The font to use for bold italic data values in the form. Defaults to `IBSO`.

`-- DataFontItalic` ⟨*font*⟩
The font to use for italic data values in the form. Defaults to `IISO`.

`-- HeadingFont` ⟨*font*⟩
The font to use for heading values in the form. Defaults to `HISO`.

`-- HeadingFontSize` ⟨*size*⟩
The size of type used in box headings. Defaults to 8.

`-- FontFace_body` ⟨*font*⟩
The font designated for use in runnint text. Defaults to `RISO`.

`-- FontFace_data_ttl` ⟨*font*⟩
The font designated for use in data and titles. Defaults to `HISO`.

`-- FontFace_footnote` ⟨*font*⟩
The font designated for use in footnotes. Defaults to `HISO`.

Color Themes

`-- SetTheme_d20_Blue --`
Defines `AltGreyHue`, `BlankHue`, `DarkHue`, `FaintHue`, `FormHue`, `GreyHue`, `HighlightHue`, and `LabelTagHue` to produce a blue color theme for the document.

`-- SetTheme_d20_Green --`
As above but sets a green theme.

`-- SetTheme_d20_Monochrome --`
As above but sets a greyscale theme.

`-- SetTheme_d20_Red --`
As above but sets a red theme.

`-- SetTheme_d20_Violet --`
As above but sets a purple theme.

`-- SetTheme_d20_Character --`
Sets up for a character record sheet. Sets the red color theme and appropriate titles for character sheets.

`-- SetTheme_d20_GM_Matrix --`
: Sets up for a `GM` party summary sheet. Sets the green color theme and appropriate titles.

`-- SetTheme_d20_Monsters --`
: Sets up for an encounter sheet. Sets the blue color theme and appropriate titles.

Housekeeping Operators

`-- PF_Continuation_Proc --`
: This is called at the top of each page, to allow the printing of some sort of header to indicate what part of the document is being continued onto this page.

Movement Operators

`-- InterBlockGap --`
: Moves down a distance appropriate to separate one data block from another. If you just start a new data block row without calling this, the new row will be directly adjacent to the previous one.

Character Sheet Operators

⟨*bg*⟩ ⟨*skin*⟩ ⟨*hair*⟩ ⟨*eyes*⟩ ⟨*app*⟩ ⟨*deity*⟩ ⟨*wt*⟩ ⟨*ht*⟩ ⟨*sz*⟩ ⟨*age*⟩ ⟨*sex*⟩ ⟨*align*⟩ ⟨*race*⟩ ⟨*fvrd*⟩ ⟨*HD*⟩ ⟨*lvl*⟩ ⟨*cls*⟩ ⟨*ID*⟩

: Set a 4-row block of cells with the indicated player and character information.

`-- BeginAbilityScores --`
: Start the ability scores, each of which is set using `AbilityScore`. This also marks our place on the page for where the content to the right of the ability scores should go.

⟨*tmpmod*⟩ ⟨*tmp*⟩ ⟨*modttl*⟩ ⟨*mod*⟩ ⟨*score*⟩ ⟨*ability*⟩ `AbilityScore --`
: Set a single ability score called ⟨*ability*⟩ with raw score ⟨*score*⟩, modifier ⟨*mod*⟩ titled ⟨*modttl*⟩, and temporary score and modifier if any. For example: `()()(STR MOD)(+0)(10)(STR)AbilityScore`

`-- EndAbilityScores --`
: End the block of ability scores.

⟨*loadspeed*⟩ ⟨*speed*⟩ ⟨*pushwt*⟩ ⟨*lift*⟩ ⟨*carry*⟩ `StrengthBlock --`
: Set the set of statistics based on strength.

`-- BeginSavingThrowBlock --`
: Start the block of saving throw stats, each set with `SavingThrow`.

⟨*misc*⟩ ⟨*race*⟩ ⟨*magic*⟩ ⟨*mod*⟩ ⟨*modname*⟩ ⟨*basemod*⟩ ⟨*score*⟩ ⟨*save*⟩ `SavingThrow --`

Set a saving throw called ⟨*save*⟩ with value ⟨*score*⟩ which is derived from the other values as shown.

⟨*mods2*⟩ ⟨*mods1*⟩ `EndSavingThrowBlock --`
End the block of saving throws. This includes setting two rows of arbitrary text describing the modifiers that may be given.

`-- BeginXPBlock --`
Start the side block which holds `XP` and other misc. information.

⟨*gp*⟩ ⟨*heropoints*⟩ ⟨*xpneeded*⟩ ⟨*xp*⟩ `XPBlock --`
Set the characters current experience points, the number needed for their next level, hero points, and gold.

`-- EndXPBlock --`
End the `XP` block.

`-- BeginAttackBlock --`
Start the set of attack types, each set by `AttackBlock`.

⟨*temp*⟩ ⟨*misc*⟩ ⟨*score2*⟩ ⟨*ttl2*⟩ ⟨*score1*⟩ ⟨*ttl1*⟩ ⟨*bab*⟩ ⟨*score*⟩ ⟨*type*⟩ `AttackBlock --`

Adds an attack line to the character sheet called ⟨*type*⟩ (usually "`Melee`", "`Ranged`", or "`CMB`"), the total attack ⟨*score*⟩ along with the components from which it was derived.

⟨*bow/sling*⟩ ⟨*two-handed*⟩ ⟨*off-hand*⟩ ⟨*normal*⟩ `DamageBlock --`
Set the damage block.

⟨*mod3*⟩ ⟨*ttl3*⟩ ⟨*mod2*⟩ ⟨*ttl2*⟩ ⟨*mod1*⟩ ⟨*ttl1*⟩ ⟨*mod0*⟩ ⟨*ttl0*⟩ ⟨*initmod*⟩ `InitiativeBlock --`

Set the initiative block with total initiative score ⟨*initmod*⟩ and up to four components from which it was derived.

`-- EndAttackBlock --`
End the attack section.

`-- BeginWeaponBlock --`
Start an arbitrary-length list of weapons, each of which is set with `Weapon`.

⟨*ID*⟩ ⟨*dtyp*⟩ ⟨*sz*⟩ ⟨*wt*⟩ ⟨*rch*⟩ ⟨*rg*⟩ ⟨*crit*⟩ ⟨*th*⟩ ⟨*dmg*⟩ ⟨*att*⟩ ⟨*desc*⟩ `Weapon --`

Add a weapon or attack form to the list.

`-- EndWeaponBlock --`
End the list of weapons.

`-- BeginDefenseBlock --`
: Start the defense area of the character sheet.

⟨*mod7*⟩ ⟨*ttl7*⟩ ⟨*dark7*⟩ ⟨*...*⟩ ⟨*mod0*⟩ ⟨*ttl0*⟩ ⟨*dark0*⟩ ⟨*score*⟩ ⟨*type*⟩ `ACBlock --`

: Sets an `AC` line called `type` (usually "`AC`", "`Touch`", "`Flat`", or "`CMD`"), its total ⟨*score*⟩ and eight different components from which the score is derived. For each of these, if ⟨*dark*⟩ is true, the box is darkened (to indicate it does not apply).

⟨*qty*⟩ ⟨*used*⟩ ⟨*lethal*⟩ ⟨*nonlethal*⟩ ⟨*sr*⟩ ⟨*dr*⟩ ⟨*healrate*⟩ ⟨*hp*⟩ `HP --`
: Set the hit point block with the given ⟨*hp*⟩ total as well as the given values for healing rate, `DR`, and `SR`, all of which are string values. The following parameters are integers: ⟨*qty*⟩ specifies how many total boxes to include on the form to check off wounds as they happen, ⟨*used*⟩ specifies the number of boxes are in play (the total hit points for the character), ⟨*lethal*⟩ is the number of hit points lost to lethal wounds, and ⟨*nonlethal*⟩ is the number of hit points in nonlethal damage.

`-- EndDefenseBlock --`
: End the defensive section.

`-- BeginArmourBlock --`
: Start a list of an arbitrary number of armors and other protective devices, each of which is set with `Armour`.

⟨*ID*⟩ ⟨*wt*⟩ ⟨*spd*⟩ ⟨*spl*⟩ ⟨*chk*⟩ ⟨*mdex*⟩ ⟨*type*⟩ ⟨*bonus*⟩ ⟨*description*⟩ ⟨*on*⟩ `Armour --`

: Add another armor to the list with the attributes listed. All are strings except ⟨*on*⟩, which is an integer: if 1, a box is checked indicating that the item is worn or active; if 0, the box is not checked.

`-- EndArmourBlock --`
: End the block of armors.

`-- BeginFeatBlock --`
: Start a list of feats and other special abilities, each set with `Feat`.

⟨*ref*⟩ ⟨*description*⟩ `Feat --`
: Set an entry in the feat list. The block has two columns; each of these sets an entry in the left column, then the right, then the next row's left, and so no.

`-- EndFeatBlock --`
: End the block of feats, adding a footer line explaining symbols used in the block.

`-- EndFeatBlockNoFooter --`
: As `EndFeatBlock` but does not include the footer at the bottom.

⟨*perlvl*⟩ ⟨*maxranks*⟩ ⟨*racial*⟩ ⟨*intmod*⟩ ⟨*base*⟩ ⟨*pts/lvl*⟩ ⟨*ttl*⟩`-- BeginSkillBlock --`

Starts a skill block, with the given values in the header of the block.

⟨*desc*⟩ ⟨*misc*⟩ ⟨*item*⟩ ⟨*race*⟩ ⟨*syn*⟩ ⟨*cls*⟩ ⟨*rnks*⟩ ⟨*mod*⟩ ⟨*abil*⟩ ⟨*ttl*⟩ ⟨*name*⟩ ⟨*a*⟩ ⟨*t*⟩ ⟨*c*⟩ `Skill --`

Add the named skill to the list of skills with the given attributes. All are strings except the three integer values on the top of the stack: ⟨*c*⟩ indicates if the skill is a class skill, ⟨*t*⟩ indicates if training is required, and ⟨*a*⟩ indicates if an armor check penalty applies; in each case a 1 indicates that the corresponding box should be checked and 0 means that it should not be.

`-- EndSkillBlock --`
Ends the skill block section.

⟨*title*⟩ `BeginAnimalBlock --`
Start describing an animal companion or familiar with the given ⟨*title*⟩.

`-- EndAnimalBlock --`
End the animal description section.

`-- BeginRelationshipBlock --`
Start the block describing the PC's relationship with NPCs.

⟨*effect*⟩ ⟨*level*⟩ ⟨*score*⟩ ⟨*npcname*⟩ `RelationshipScore --`
Add a relationship to the block.

`-- EndRelationshipBlock --`
End the list of relationship scores.

⟨*wt*⟩ `BeginItemBlock --`
Start the inventory section, indicating the total weight carried as ⟨*wt*⟩.

⟨*ID*⟩ ⟨*wt*⟩ ⟨*location*⟩ ⟨*description*⟩ ⟨*total*⟩ ⟨*used*⟩ ⟨*checked*⟩ ⟨*half*⟩ `Item --`

Add an item to the inventory with the given attributes. All are strings except the following: ⟨*total*⟩ If nonzero, this item entry actually describes a quantity of consumable items, represented as boxes on the character sheet. The total number of boxes is ⟨*total*⟩, ⟨*used*⟩ is the number of boxes in use, ⟨*checked*⟩ is the number to put full crosses over, and ⟨*half*⟩ is the number to half-check.

`-- EndItemBlock --`
End the inventory section.

⟨*ttl9*⟩ ⟨*av9*⟩ ⟨*cst9*⟩ ⟨*prp9*⟩ ⟨*nrm9*⟩ ⟨*dcst9*⟩ ⟨*dprp9*⟩ ⟨...⟩ ⟨*ttl0*⟩ ⟨*av0*⟩ ⟨*cst0*⟩ ⟨*prp0*⟩ ⟨*nrm0*⟩ ⟨*dcst0*

Start a spell slot block with the given ⟨*title*⟩. The values ⟨*know0*⟩

through ⟨*know9*⟩ are strings holding the number of spells known at each level, if applicable. The other values are integers and are repeated ten times to give the values for spell level IX through 0. ⟨*Total*⟩ is the total number of spell slot boxes to draw, ⟨*avail*⟩ is the number that are actually available, ⟨*cast*⟩ is the number already cast, ⟨*prep*⟩ is the number prepared, ⟨*normal*⟩ is the number of these boxes that are normal (as opposed to domain, etc.) spells, ⟨*dcast*⟩ is the number of special slots already cast, and ⟨*dprep*⟩ is the number of special slots already prepared.

⟨*DC9*⟩ ⟨*...*⟩ ⟨*DC0*⟩ ⟨*concentration*⟩ ⟨*title*⟩ `BeginSpellBlock --`
Start a block of spell descriptions. All parameters are strings. The ten values ⟨*DC9*⟩ through ⟨*DC0*⟩ give the saving throw DC for spell levels IX to 0 respectively.

⟨*ref*⟩ ⟨*sr*⟩ ⟨*sv*⟩ ⟨*dur*⟩ ⟨*trg*⟩ ⟨*rng*⟩ ⟨*comp*⟩ ⟨*ct*⟩ ⟨*sch*⟩ ⟨*name*⟩ ⟨*lvl*⟩ ⟨*prep*⟩ ⟨*cast*⟩ ⟨*flagarr*⟩ ⟨*textarr*⟩ `Spell --`

Describe a spell with the given attributes (all strings except the following). ⟨*Prep*⟩ is the number of instances of this spell prepared, ⟨*cast*⟩ is the number actually cast, and ⟨*flagarr*⟩ is an array of flags to write over the checkboxes used to indicate prepared/cast spells, to indicate metamagic aspects used when preparing them. The ⟨*textarr*⟩ array is in the form expected by `WrapAndFormat`.

The operator `SpellInstanceExtension` is executed before setting the main spell description text. Define this to any code you need to include in order to write additional information about the spell, including metamagic-altered spells prepared using nontrivial checkbox labels. `SpellInstanceExtension` will then be reset to an empty procedure by `Spell`.

`-- EndSpellBlock --`
End the block of spell descriptions started by `BeginSpellBlock`.

⟨*maxpool*⟩ ⟨*daily*⟩ ⟨*title*⟩ `BeginExploitBlock --`
Start a list of arcanist exploits. (⟨*title*⟩ may not actually be implemented.)

⟨*save*⟩ ⟨*dur*⟩ ⟨*trg*⟩ ⟨*rng*⟩ ⟨*cast*⟩ ⟨*DC*⟩ ⟨*name*⟩ ⟨*type*⟩ ⟨*cost*⟩ ⟨*textarr*⟩ `Exploit --`

Add an exploit to the list in a manner analogous to `Spell`.

`-- EndExploitBlock --`
End the list of exploits.

Monster Encounter Support Operators

⟨*ID*⟩ ⟨*title*⟩`-- BeginModule --`
Start the encounter document for the module with the given ⟨*title*⟩ and ⟨*ID*⟩.

⟨*title*⟩ ⟨*ID*⟩ `BeginArea --`
Start a new encounter area with the given ⟨*ID*⟩ and ⟨*title*⟩.

`-- ReserveMonsterSpace --`
Ejects the page if we don't have enough space to format a typical monster block.

`-- ReserveMinimalMonsterSpace --`
As above but assumes a very terse monster description.

⟨*xp*⟩ ⟨*cr*⟩ ⟨*ref*⟩ ⟨*align*⟩ ⟨*imagetag*⟩ ⟨*type*⟩ ⟨*species*⟩ ⟨*genericname*⟩ `BeginMonster --`

Start a new monster listing with the given string attributes.

⟨*name*⟩ `TemplateDeclaration --`
Indicate that the named template was applied to the creature.

⟨*array*⟩ `MonsterDescription --`
Typeset the monster's physical appearance. The ⟨*array*⟩ is as expected for `LimitedFreeFormTextBlock`.

⟨*cmdtxt*⟩ ⟨*cmd*⟩ ⟨*cmbtxt*⟩ ⟨*cmb*⟩ ⟨*spd*⟩ ⟨*HD*⟩ ⟨*rchtxt*⟩ ⟨*rch*⟩ ⟨*spctxt*⟩ ⟨*spc*⟩ ⟨*sz*⟩ ⟨*dexmod*⟩ ⟨*dex*⟩ ⟨*i*

Set the monster's basic stats.

⟨*will*⟩ ⟨*refl*⟩ ⟨*fort*⟩ `MonsterSavingThrows --`
Set the saving throws for the creature.

`-- BeginMonsterDefenseBlock --`
Start the monster's defense area

⟨*regen*⟩ ⟨*vuln*⟩ ⟨*resist*⟩ ⟨*sr*⟩ ⟨*dr*⟩ `MonsterMiscDefenses --`
Set the monster's misc defensive stats.

`-- EndMonsterDefenseBlock --`
End the monster defense area.

`-- BeginMonsterWeaponBlock --`
Start the list of monster attack modes.

⟨*reach*⟩ ⟨*range*⟩ ⟨*crit*⟩ ⟨*threat*⟩ ⟨*damage*⟩ ⟨*attack*⟩ ⟨*description*⟩ ⟨*isalt*⟩ ⟨*isrange*⟩ `MonsterWeapon`

If ⟨*isrange*⟩ is true, set this with a darker background to indicate a ranged weapon. Otherwise, print in normal color or alternate color depending on ⟨*isalt*⟩.

`-- EndMonsterWeaponBlock --`
End the list of monster attacks.

⟨*chmd*⟩ ⟨*wsmd*⟩ ⟨*inmd*⟩ ⟨*cnmd*⟩ ⟨*dxmd*⟩ ⟨*stmd*⟩ ⟨*cha*⟩ ⟨*wis*⟩ ⟨*int*⟩ ⟨*con*⟩ ⟨*dex*⟩ ⟨*str*⟩ `MonsterAbilityScores`

Set the monster's ability scores and their modifiers (all strings).

`-- BeginMonsterSkills --`
Start the block of monster skills.

⟨*mod*⟩ ⟨*skill*⟩ `MonsterSkill --`
Set each skill.

`-- EndMonsterSkills --`
End the skill block.

`-- BeginMonsterInstanceBlock --`
Start the list of instances of individual monsters."

⟨*totalhp*⟩ ⟨*lethal*⟩ ⟨*nonlethal*⟩ `MonsterInstance --`
Set an individual monster instance.

`-- EndMonsterInstanceBlock --`
End the list of individuals.

`-- EndMonster --`
End the monster entry.

`-- EndArea --`
End the encounter area.

`-- EndModule --`
End the encounter document.

Skills Matrix

`-- BeginStatMatrix --`
Start a matrix of party stats for the GM's reference.

`-- MXsBeginBasic --`
Start the matrix, printing headings.

`-- SM_EvenRow --`
Start writing an even-numbered row

`-- SM_OddRow --`
Start writing an odd-numbered row

⟨*dc*⟩ ⟨*cmd*⟩ ⟨*flat*⟩ ⟨*tch*⟩ ⟨*ac*⟩ ⟨*init*⟩ ⟨*imod*⟩ ⟨*chmd*⟩ ⟨*cha*⟩ ⟨*wsmd*⟩ ⟨*wis*⟩ ⟨*inmd*⟩ ⟨*int*⟩ ⟨*comd*⟩ ⟨*con*⟩ ⟨*dxmd*⟩

Write a row of the character stat matrix.

`-- EndStatMatrix --`
End the stats matrix.

`-- BeginSkillMatrixTitle --`
: Start the title block of the skill matrix.

⟨*name*⟩ `SMSkillName --`
: Add the named skill as the next column of the matrix.

`-- EndSkillMatrixTitle --`
: End the title line of the matrix.

⟨*name*⟩ `BeginPlayerSkillBlock --`
: Start the skill block.

⟨*mod*⟩ `SMSkillValue --`
: Add another value column to the matrix.

`-- SMSkillNull --`
: Add a nulled-out column to the matrix.

`-- EndPlayerSkillBlock --`
: End the player's block

`-- EndSkillMatrix --`
: End the skill matrix.

`-- BeginSkillMatrix --`
: Start a matrix of party skill for the GM's reference.

Wrapped Text Support

⟨*array*⟩ `FreeFormTextBlock --`
: Typeset the text chunks in ⟨*array*⟩ as the lower-level operator `WrapAndFormat` would, at the full page width.

⟨*array*⟩ ⟨*w*⟩ `LimitedFreeFormTextBlock --`
: As above but in a limited space with width ⟨*w*⟩.

⟨*title*⟩ `DividerBlock --`
: Set a full-width section divider with the given ⟨*title*⟩.

⟨*array*⟩ `NotesBlock --`
: Set the ⟨*array*⟩ as `FreeFormTextBlock` in a section called "Notes".

The following operators are available for use with the text arrays passed to `WrapAndFormat`:

⟨*level*⟩ `PsFF_ind --`
: Set indentation level and move current point to start of the line at that position.

`-- PsFF_nl --`
: Start a new line.

⟨*str*⟩ ⟨*w*⟩ `PsFF_t`⟨*t*⟩⟨*a*⟩ `--`
: Typeset a table cell with string ⟨*str*⟩ and width ⟨*w*⟩. In the name of the operator, ⟨*t*⟩ can be `h` to set a header cell or `d` for a data cell; ⟨*a*⟩ can be `C` for centered data, `L` for left-justified, or `R` for right-justified text.

Miscellaneous

`-- StartLandscape --`
: Switch to landscape orientation.

`-- EndLandscape --`
: Switch to portrait orientation.

SEE ALSO

`form-preamble`(7).

AUTHOR

Steve Willoughby / steve@madscience.zone.

HISTORY

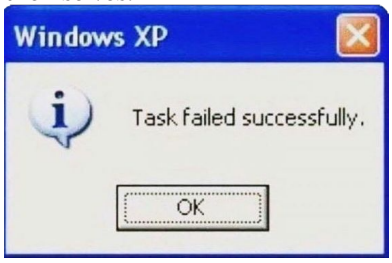
COPYRGHT

Part of the GMA software suite, copyright © 1992–2025 by Steven L. Willoughby, Aloha, Oregon, USA. All Rights Reserved. Distributed under BSD-3-Clause License.

APPENDIX G

How to Install the Mapper on Windows 10

When you cast *fireball* at a wild mage, they counter-spell it but they get a wild surge which makes them cast *fireball* on themselves.



—Internet meme

This tutorial will walk you through the process of setting up a GMA Mapper client on Windows 10. Once set up, refer to the full tutorial on the mapper client. (See p. 725.)

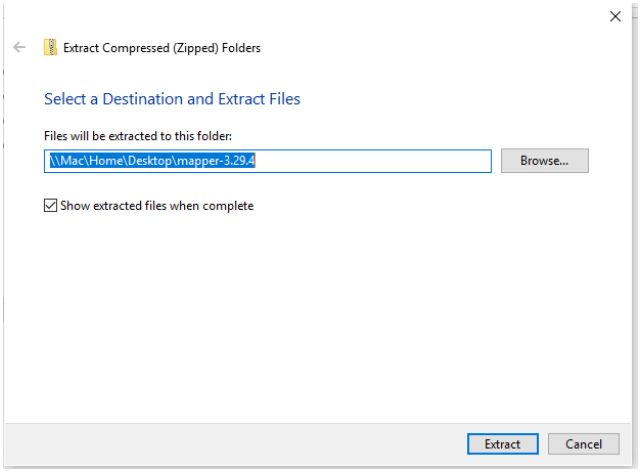
STEP 1: Obtain and Unpack the Mapper

You need to download and unpack a copy of the GMA Mapper program before proceeding on to the next step.

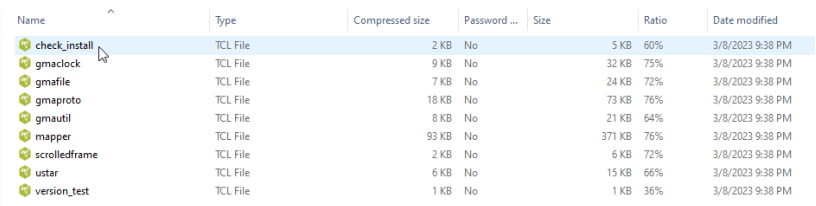
Once you install it the first time, you can repeat this process to install a newer version later, or if your GM has set up their own game server appropriately, the mapper will automatically upgrade itself whenever your GM makes a newer version available to your group.

(A) Download the mapper client as a single ZIP archive file from github.com/MadScienceZone/gma-mapper/tree/main/signed-releases/.

(B) Unpack it in some convenient folder, using a ZIP extraction program or the built-in Windows feature to extract files from the ZIP file into a folder of your choice.



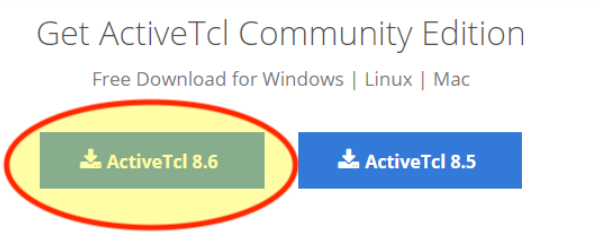
(C) Click into that folder, then into the `bin` subfolder. You should see a collection of files like this:



STEP 2: Install Tcl/Tk

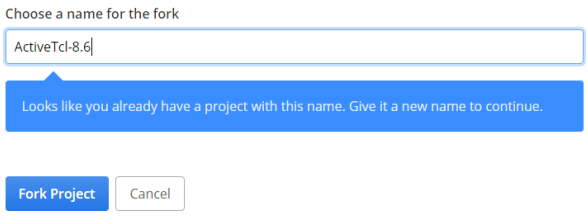
The GMA Mapper is written in the Tcl/Tk scripting language, so you'll need an interpreter for that language on your system. If you don't already have one, follow these steps to get one:

(A) Download the installer from ActiveState's site. Click on the newest version offered (at least 8.6 is required).

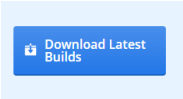


a. You will be prompted to log in to your account or create a new free one.

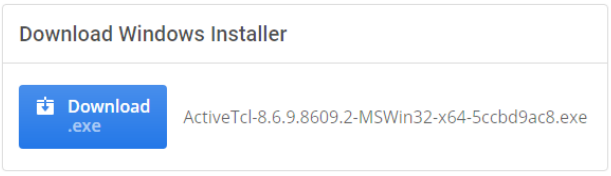
b. Next, you'll be asked to fork the Tcl project into your account so you have your own private copy. Give it a name and click Fork Project.



c. Click Download Latest Builds.



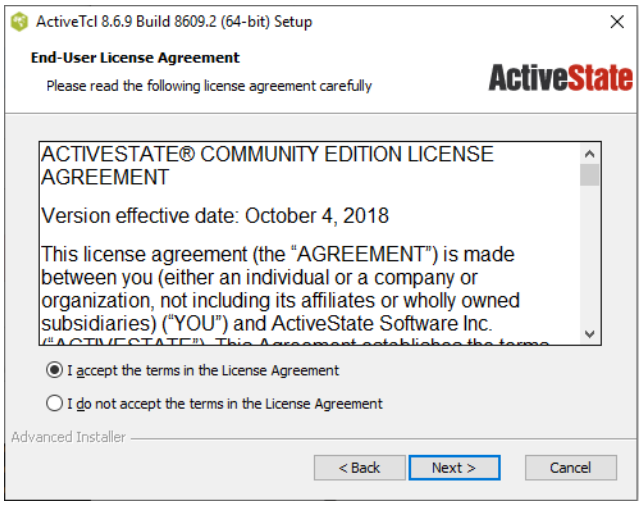
d. Click Download .exe



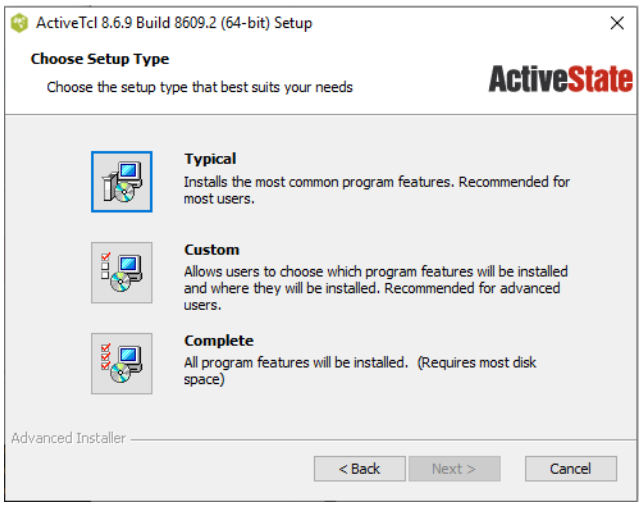
e. Run the installer file you just downloaded. Click Next >.



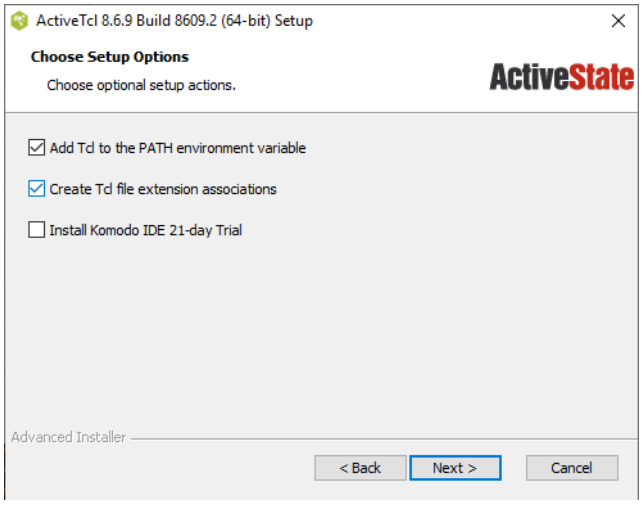
f. If you agree with the EULA, click the appropriate button and then Next >.



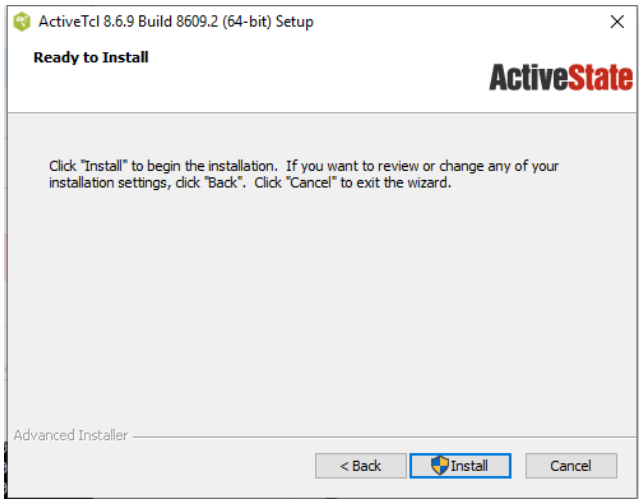
g. Choosing the typical install should be sufficient, but you may install a more complete package if you like.



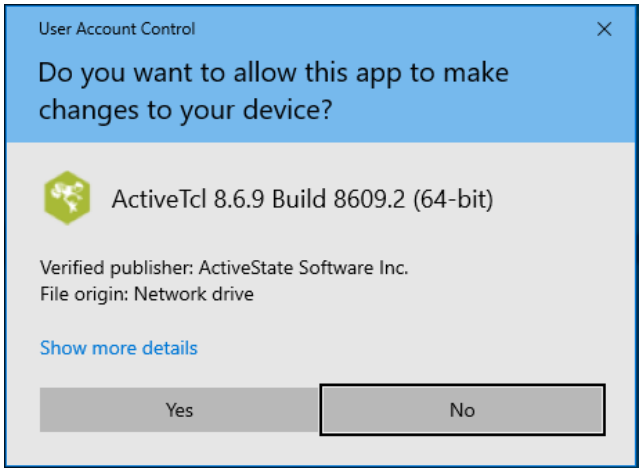
h. Check these boxes and click Next >.



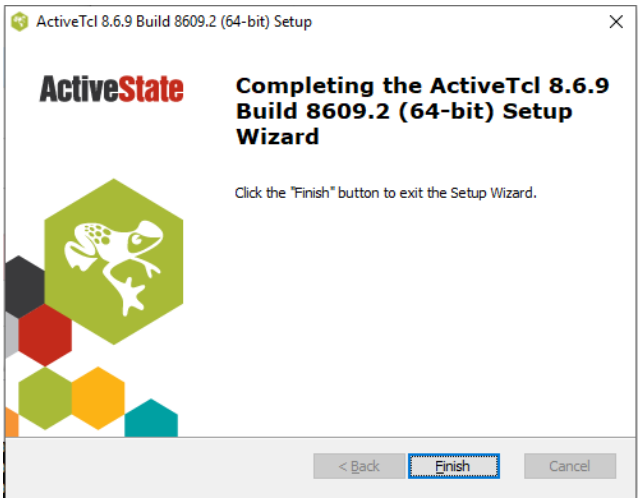
i. Click Install.



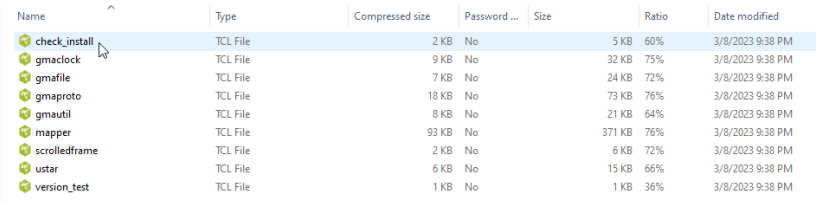
j. Allow the installer to install.



k. When the installer is finished, click Finish.

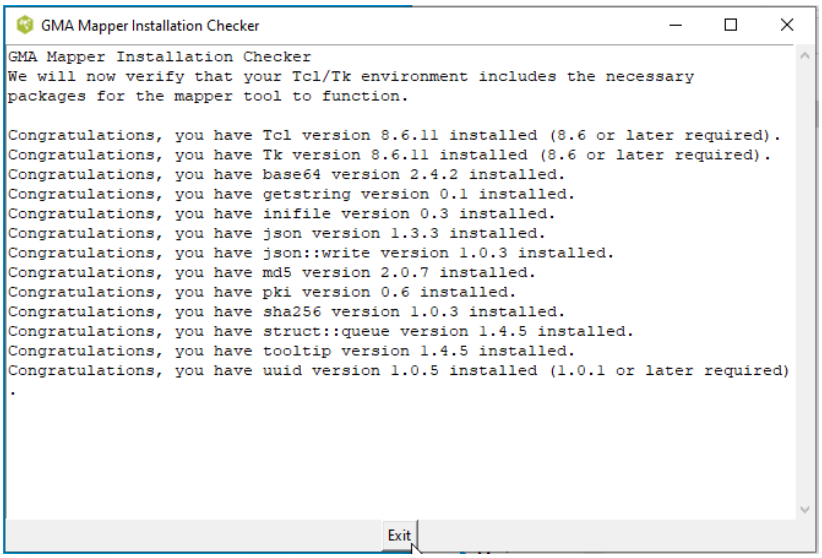


(B) Verify that everything's installed ok. Go back to the `bin` subfolder inside the location where you unpacked the GMA Mapper files. Double-click on the `check_install` file.



Name	Type	Compressed size	Password ...	Size	Ratio	Date modified
check_install	TCL File	2 KB	No	5 KB	60%	3/8/2023 9:38 PM
gmaclock	TCL File	9 KB	No	32 KB	75%	3/8/2023 9:38 PM
gmafile	TCL File	7 KB	No	24 KB	72%	3/8/2023 9:38 PM
gmaproto	TCL File	18 KB	No	73 KB	76%	3/8/2023 9:38 PM
gmautil	TCL File	8 KB	No	21 KB	64%	3/8/2023 9:38 PM
mapper	TCL File	93 KB	No	371 KB	76%	3/8/2023 9:38 PM
scrolledframe	TCL File	2 KB	No	6 KB	72%	3/8/2023 9:38 PM
ustar	TCL File	6 KB	No	15 KB	66%	3/8/2023 9:38 PM
version_test	TCL File	1 KB	No	1 KB	36%	3/8/2023 9:38 PM

If all goes well, you should see a window which tells you that you have all of the required library files:

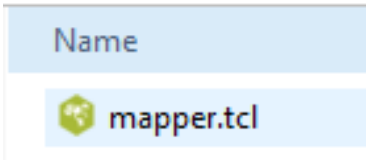


If you see messages that say you're missing any required libraries, follow the advice shown in the window. This should not happen, however, because ActiveState Tcl comes out of the box with everything you should need. Generally, if something is missing it will be the library `tklib`. Refer to the ActiveState Tcl site for help adding this library to your system if your copy of Tcl/Tk didn't come with it.

Click the Exit button.

(C) While still in the subfolder where you clicked on `check_install`, double click on the `mapper` program.

(D) Launch the Mapper application.



(E) You'll see the map and can try out all the local map-drawing stuff mentioned in the documentation and tutorial on page 725.

STEP 3: Configure Mapper for Server Access

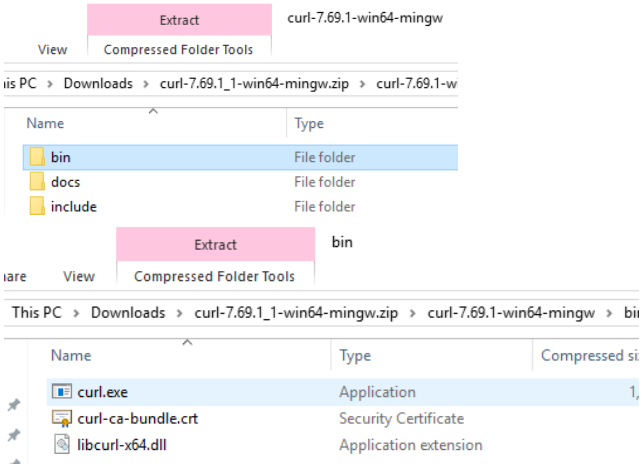
If you want to use the mapper with a server so it's connected to the GM's mapper and the mappers of other players in a group, you'll need to do the following.

(A) Install the Curl program. (You might not actually need to do this. Windows 10 comes with a version of Curl built-in that may suffice for our purposes. Look for one in your system's standard `%SYSTEM32%` folder. If there's a working copy there, just use that—i.e., configure its pathname into the mapper configuration file as described below rather than installing a new Curl program).

a. Download a copy from https://curl.haxx.se/windows/. Click on the 32- or 64-bit version as appropriate for your system.



b. Unpack the ZIP file you just downloaded into a convenient location, such as `C:\curl-7.67.0-win64-mingw`.



(B) Configure the Mapper client. Start the mapper client by double-clicking on the `mapper.tcl` file icon as we did previously. From the main menu, choose Edit ⟩ Preferences... as described fully starting on page 726.

Now start the mapper client and it should connect to the game server, interacting with the game you're playing with your friends.

APPENDIX H

How to Install the Mapper on Linux, BSD, and macOS

> By the way, if you get mad at your Mac laptop and wonder who designed this demonic device, notice the manufacturer's icon on top: an apple with a bite out of it.
>
> —Peter Kreeft

For various reasons—either lost to the mists of time and legend, or believed by many that it would be better if they were—installing GMA on a macOS system is a tricky prospect, but have no fear. This chapter will walk you through the process of setting up a GMA Mapper client on such a system.

Installing on other Unix-like operating systems such as Linux or FreeBSD is not as fraught with caveats and pitfalls,[1] follows a very similar process, so we will cover those platforms here as well.

Once set up, refer to the full tutorial on using the mapper client. (See p. 725.)

[1] Mostly due to the lack of a frequently-out-of-date vendor-supplied version of Tcl/Tk.

STEP 1: Install Homebrew (Mac only)

Installing and maintaining open source software on a macOS system is much easier if you use a package management system. There are several available for macOS, but the one we will recommend here is Homebrew.

To install Homebrew on your system, see the instructions posted on their website at https://brew.sh.

As of this writing, that involves running the command[2]

```
$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.co
m/Homebrew/install/HEAD/install.sh)"
```

Linux and BSD systems should already have a package manager installed out of the box, although which one, and the particulars for how to use it, will vary slightly depending on the particular Linux distro or flavor of BSD, but will usually be something like `pkg`, `apt`, `yum`, `pacman`, etc.

STEP 2: Install GMA Mapper

Easy Version

The easy way to do this is to unpack a ready-to-run copy of the mapper.

If your GM maintains a server for your campaign, they may make their chosen version of the GMA Mapper available to your group on that server. In this case, your installed copy of the mapper will notice when the GM has updated the version of the mapper they want you to use, and it will automatically upgrade itself when that happens.

Otherwise, you will need to repeat this step in the future to download and install an updated version of the mapper as you wish.

(A) Download the mapper client as a single ZIP or `tar` archive file from github.com/MadScience.Zone/gma-mapper/tree/main/signed-releases.

(B) Unpack it in some convenient folder.

```
$ cd my_gma_files
$ tar xvfpz mapper-3.40.6.tar.gz
```

(C) `cd` into the `bin` subdirectory. You should see a number of Tcl source files including `check_install.tcl` and `mapper.tcl`.

[2]The more security-minded will balk at this since it blindly downloads a shell script from the Internet and executes it. You can, of course, use `curl` or other program to download the `install.sh` file, inspect it, and then execute it when you're satisfied it won't do anything evil.

Advanced Version

For the advanced user, rather than downloading and unpacking an archive file, make sure Git is installed on your system and pull GMA from the repository https://github.com/MadScienceZone/gma-mapper. You can run the mapper client directly from the `bin` subdirectory of the cloned repository.

The client will not automatically upgrade itself if installed this way, but you can keep the client up to date by running `git pull`. Using Git also allows you to run experimental code from the `develop` branch and to contribute to the development of the GMA Mapper software if you desire.

For example:

```
$ git clone https://github.com/MadScienceZone/gma-mapper
Cloning into 'gma-mapper'...
remote: Enumerating objects: 2146, done.
remote: Counting objects: 100% (551/551), done.
remote: Compressing objects: 100% (227/227), done.
remote: Total 2146 (delta 314), reused 537 (delta 300), pack-
reused 1595
Receiving objects: 100% (2146/2146), 15.06 MiB | 5.66 MiB/s,
done.
Resolving deltas: 100% (1084/1084), done.
$ cd gma-mapper/bin
$ ls
check_install.tcl  gmautil.tcl            ustar.tcl
gmaclock.tcl       mapper.tcl             version_test.tcl
gmafile.tcl        scrolledframe.tcl
gmaproto.tcl       test_clock_widget.tcl
```

Additional Notes

If you also installed the full GMA Core software suite, you may need to set the environment variable `GMA_WISH` to the location of the Tcl/Tk interpreter you installed, `GMA_MAPPER` to the location where you unpacked the mapper client, and the `GMA` variable to where you installed GMA. For example:

```
$ export GMA=/Users/steve/gma
$ export GMA_MAPPER=/Users/steve/gma-mapper
$ export GMA_WISH=/usr/local/opt/tcl-tk/bin/wish8.6
```

Adjust the pathnames from the example above for the directories where you installed GMA Core, Mapper, and Tcl/Tk on your system.

STEP 3: Install Tcl/Tk

macOS

The Apple versions of Python and Tcl are not usually the most up-to-date, and historically their associated Tk libraries are increasingly incompatible with changes released to the macOS GUI libraries they depend on.[3]

As such, it's important that you install the latest public versions of these tools via Homebrew, and ensure that GMA uses those instead of the built-in ones.

This can be done by issuing the command:

```
$ brew install tcl-tk
```

Note that this will install the new Tcl/Tk code in a new location (separate from the one that came with the operating system), such as `/usr/local/opt/tcl-tk`. For the instructions that follow, we assume this to be, in fact, the installation directory for your system. If yours is installed in a different location, be sure to use the correct pathname as applicable for your system.

FreeBSD

Your system may already have Tcl/Tk pre-installed. Check this by typing the commands

```
$ which tclsh
/usr/local/bin/tclsh
$ which wish
/usr/local/bin/wish
```

It's possible that those commands don't exist on your system but a specifically-versioned command name does (or, multiple different ones if you have more than one version of Tcl/Tk installed):

[3] We are only covering the installation of Tcl/Tk in this chapter, since that's all you need to use the mapper client. If you want to use the other GUI tools in the GMA suite, you'll need to set up Python as well, which will require a little more work on the macOS platform to make sure you install an updated Python package based on the Tcl/Tk libraries you installed by following these instructions, not the Python interpreter nor its Tkinter library supplied by Apple.

```
$ which tclsh
$ which wish
$ which tclsh8.6
/usr/local/bin/tclsh8.6
$ which wish8.6
/usr/local/bin/wish8.6
$ which tclsh8.7
/usr/local/bin/tclsh8.7
$ which wish8.7
/usr/local/bin/wish8.7
```

In the above example, Tcl/Tk version 8.6 and 8.7 are installed, so you will need to specify which interpreter to use to run the GMA programs. For example:

```
$ wish8.6 check_install.tcl
```

If you don't have Tcl/Tk installed (or just have Tcl but not Tk, which would be the case if you can find `tclsh` or `tclsh8.6` but not `wish` or `wish8.6`), you will need to install them before continuing further.

```
$ sudo pkg install tcl87
$ sudo pkg install tk87
```

Then, retry the `which` commands above to verify that you have `tclsh` and `wish` installed, and if you need to use the versioned names `tclsh8.7` or `wish8.7` (or 8.6, or whatever version you installed).

Linux

Installing Tcl/Tk on Linux is mostly identical to the instructions for FreeBSD in the previous section, with the main difference being what package manager your system uses and possibly what the Tcl and Tk packages are called in that system.

On Arch, for example, using the `pacman` program, they are installed by typing

```
$ sudo pacman -S tcl tk
```

On Ubuntu, using the `apt` program, the process is similar:

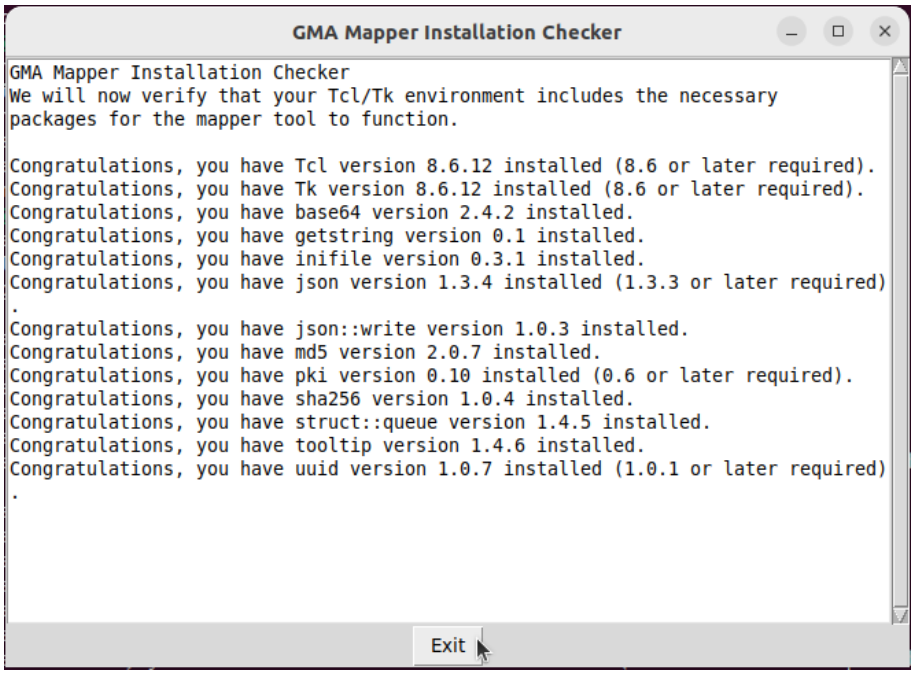
```
$ sudo apt install tcl tk
```

STEP 4: Check for Required Libraries

The GMA Mapper requires the Tcl/Tk standard libraries, which are called `tcllib` and `tklib`. Mapper comes with a script to check if those requirements are met by your system. Invoke the `bin/check_install.tcl` script (located under the directory where you installed Mapper) using the version of wish appropriate to your system. That will probably be one of the following:

```
$ cd gma-mapper/bin
$ wish check_install.tcl
$ wish8.6 check_install.tcl
$ wish8.7 check_install.tcl
```

This will display a list of the required library packages and which ones are missing. If all is as it should be, you'll see a window like this:



If `check_install.tcl` tells you that you're missing `tcllib` or `tklib`, you will need to install those before Mapper will work.

Installing from Your Package Manager

Your system may provide ready-to-install packages for those. In our testing we found that Arch and FreeBSD provide `tcllib` via the command

```
$ sudo pacman -S tcllib
```

and

```
$ sudo pkg install tcllib
```

respectively, while Ubuntu provides both libraries via the command

```
$ sudo apt install tcllib tklib
```

Doing It Yourself

If you can't get the libraries that way, you can get them from http://tcl.tk. They are also available from the repositories https://github.com/tcltk/tcllib and https://github.com/tcltk/tklib.

If you cloned the GMA Core repository, copies of both libraries are available in the `dependencies` directory there as well.

Once you have obtained each missing library from one of the above sources, and unpacked them into a directory if necessary, you can install them on your system by running the `installer.tcl` script in the top level directory of each library source tree. This needs to be run with the `wish` program you installed or found previously. Again, depending on your system that may be called `wish`, `wish8.6`, `wish86`, `wish8.7`, etc. For example:

```
$ cd tklib-0.7
$ sudo wish8.6 ./installer.tcl
```

You will see a dialog box confirming what you want to install, and to where:



STEP 5: Configure Mapper for Server Access

If you want to use the mapper with a server so it's connected to the GM's mapper and the mappers of other players in a group, you'll need to do the following.

(A) Install the Curl program. (You might not actually need to do this, since your system may already come with a version of Curl built-in that may suffice for our purposes.)

(B) Configure the Mapper client. Launch the mapper by running the `bin/mapper.tcl` program. You may need to explicitly name a `wish` interpreter to use, like this:

```
$ wish8.6 gma-mapper/bin/mapper.tcl
```

Now from the main menu, choose Edit › Preferences... and follow the instructions starting on page 726.

Now start the mapper client by running `bin/mapper.tcl` with your `wish` command and it should connect to the game server, interacting with the game you're playing with your friends.

```
$ wish8.6 gma-mapper/bin/mapper.tcl
```

(adjusting the exact `wish` interpreter name and pathname to `mapper.tcl` as necessary).

APPENDIX I

Player's Handbook for the Mapper

> Games give you a chance to excel, and if you're playing in good company you don't even mind if you lose because you had the enjoyment of the company during the course of the game.
>
> —Gary Gygax

In our Pathfinder games, we use an electronic tactical battle grid to show everyone's positions in the environment, particularly useful when in combat, since the mapper also shows threatened spaces, opponents in range of each other, and other useful information. While we display this map on a large screen for everyone to see, you may wish to have your own device showing the mapper. Besides being closer to view, this allows you to actively manipulate the map by, for example, moving your own character token around to where you want your character to be in each turn.

This tutorial will explain step-by-step how to use the GMA Mapper tool. Please note that while the mapper tool has many features of use to the GM (creating and modifying the maps themselves, adding, removing, or changing the statuses of monsters, or even invoking commands that affect other connected clients' operation), this tutorial will *only* explain those features intended for players. All users will also want to read Chapter 10 for information about using the mapper's die-rolling and chat message capabilities.

GMs should also read chapters 4, 5, 9, and 11 to get the full story of the mapper's features, most of which are mostly of interest for the GM setting up the map for upcoming game sessions.

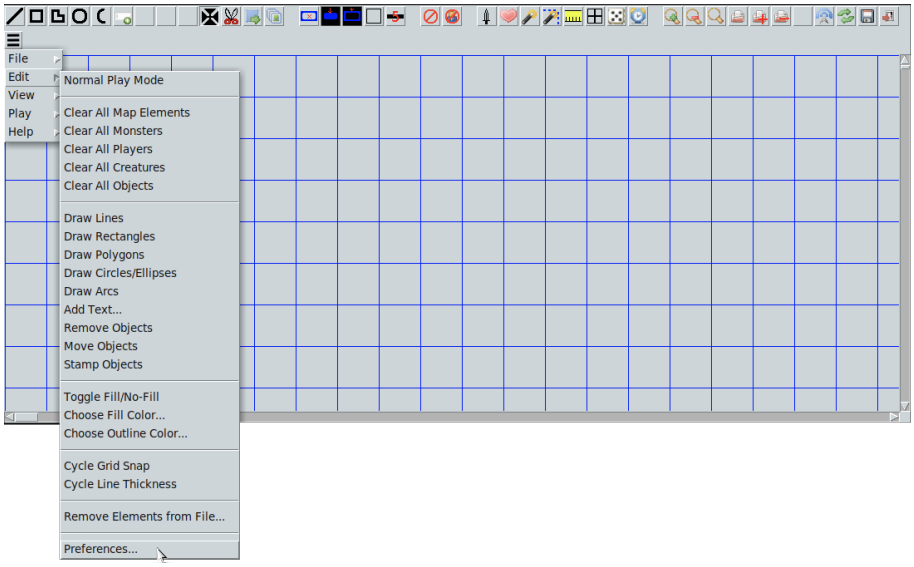
This documentation describes GMA Mapper version 4.33.1.

Installing the Mapper Client

Detailed instructions for installing the client on Windows and Unix-like systems (including Linux, FreeBSD, and macOS systems) can be found in Appendices G (page 707) and H (page 717), respectively.

Configuring the Mapper

While there are many command-line options to let you alter the operation of the mapper for each invocation (as documented starting on p. 470), the easiest way to set it up is to set up a set of persistent configuration options that the mapper will use each time it is started. This is most easily done by selecting Edit » Preferences... from the main menu.



This brings up the editor window, with several tabs holding different categories of settings. All of them, except the "Servers" tab, affect all invocations of the mapper. Let's look at each of those first.

Appearance Tab

The "Appearance" tab (Figure I.1) controls the general mapper look and feel. This includes the following settings:

Animate updates If checked, map elements such as lines, polygons, etc, are added one at a time to the map surface so you can see the map build incrementally. (Same as the `--animate` CLI option.)

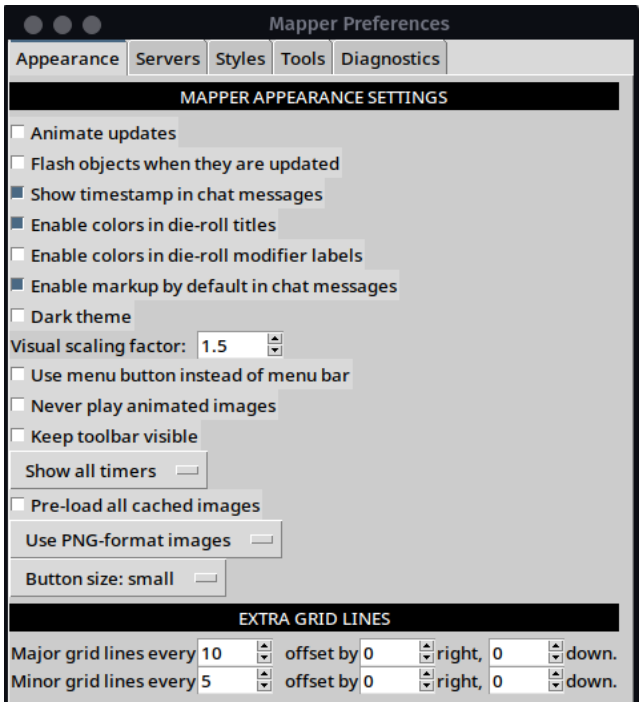


Figure I.1: Appearance Tab

Flash objects when they are updated If checked, objects on the map will flash briefly when the mapper receives an update for them, such as updated conditions or hit points.

Show timestamp in chat messages If checked, the time when each chat message and die-roll result was sent is printed next to the message itself.

Enable colors in die-roll titles If checked, die-roll titles will be displayed in full-color if they were rolled as such. Otherwise, they will be displayed without colorization.

Enable colors in die-roll modifier labels If checked, you can also colorize the labels you place on individual die-roll modifiers in the middle of die roll expressions.

Enable markup by default in chat messages If checked, this checks the "markup" (M) box in the chat window to enable markup formatting codes in chat messages that you type, so it's on when the mapper starts. Otherwise, you'll need to turn it on when you want to use formatting codes each time.

Dark theme Selects the dark color theme. (Same as the `--dark` CLI option.)

Visual scaling factor Scales up GUI elements by the given amount. This can compensate for the effects of very high resolution displays.

Use menu button instead of menu bar If checked, the main menu will pop out of a small "hamburger" button in the upper-left corner. Otherwise the application will have a menu bar across the top of the main window.

Never play animated images If checked, animated elements such as dungeon tiles and creature tokens will not be played. They will remain static and unmoving.

Keep toolbar visible Causes the mapper to ignore GM instructions to remove toolbars from all player mapper clients to increase the visible space on the map surface. If checked, your toolbar will always be displayed. (Same as the `--keep-tools` CLI option.)

Show only my timers Controls the default timer visibility on the initiative clock display. This is a menu which allows for the default to be to show no timers, only those specifically targetted at you (assuming you are signed on to the mapper with the same name—including capitalization—as the name of your PC on the map). This is the default when the mapper is started. During runtime, the visibility may be changed via the View » Show Timers menu.

Pre-load all cached images When checked, the mapper will load into its memory all of the images it has cached at start-up. This increases the speed of displayed images at the expense of runtime memory. (Same as the `--preload` CLI option.)

Use PNG format images Selects whether to use PNG or GIF image format for images loaded from the server and used as tool icons. (Same as the `--image-format` CLI option.)

Button size Selects whether you wish to use small, medium, or large-size toolbar buttons. (Same as the `--button-size` CLI option.)

Major/Minor grid lines Normally, the mapper draws grid lines every 5 feet on the mapper surface. By entering nonzero numbers here you can add "major" grid lines (which are extra-thick and in a different color) at every *n* grid lines, where *n* is the number entered in the first box. For example, if you ask for major grid lines every 5, then the 0th, 5th, 10th, 15th, etc., lines will be drawn in the major grid line style. If you don't want to start them at the far top and/or left of the map, offsets may be specified to start counting some number of grid lines to the right and/or down. (Same as the `--major` CLI option.)

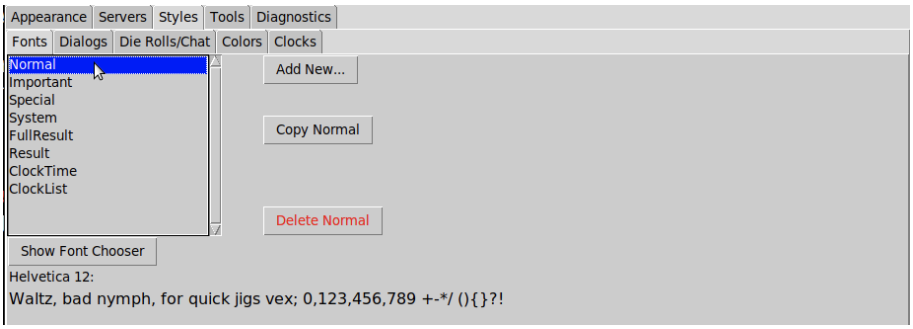


Figure I.2: Fonts Tab

"Minor" grid lines work the same, but they are in yet another color and have a thickness between the major grids and the normal ones. (Same as the `--guide` CLI option.)

Styles Tab

The "Styles" tab provides the ability to control the visual appearance of various parts of the user interface. (Note that not all of the mapper tool is under the control of the preferences editor, but we're working on that over time. For now, this mostly applies to some pop-up windows and the chat/die-roll window.)

Styles: Fonts Tab

The Styles tab (Figure I.2) has five tabs of its own. The first of these allows you to define the fonts used by the program. These are user-defined fonts which will be used on the other tabs. Clicking on the name of one of the fonts in the list selects it as the font you are altering here. In the example above, a font called "Normal" is selected. The four buttons on this tab have the following functions:

Add New... Adds a new font to the list with default attributes.

Copy ⟨***fontname***⟩ Adds a new font to the list which is a copy of the selected font.

Delete ⟨***fontname***⟩ Removes the selected font from the list. You can't do this if the font is in use by one of the style configurations.

Show Font Chooser Displays the font chooser dialog box as shown in Figure I.3. Use this dialog to select the desired font. Clicking OK will set that as the appearance of the selected font in the list.

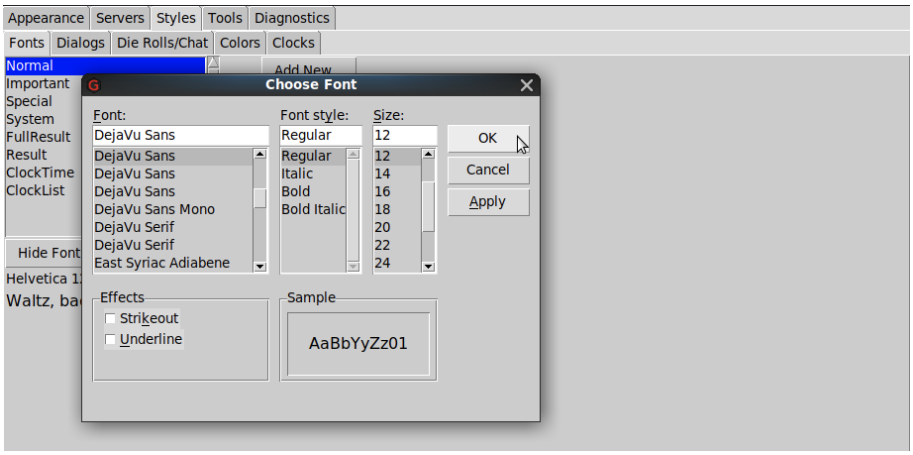
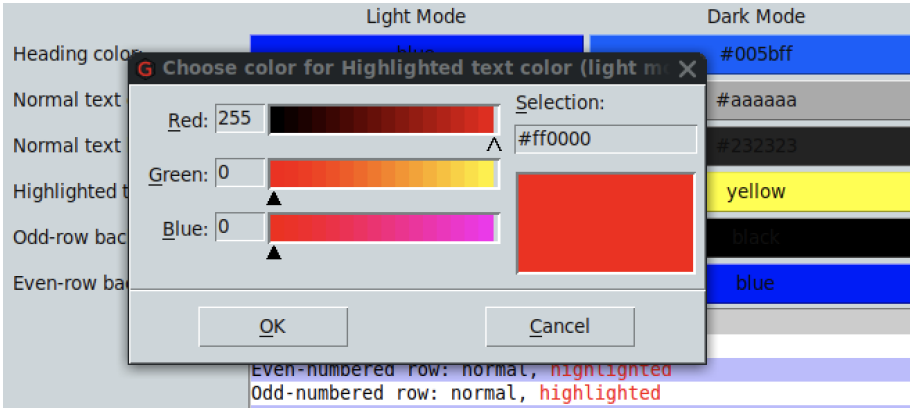


Figure I.3: Font Selection

Styles: Dialogs Tab

The "Dialogs" tab (Figure I.4) controls the appearance of pop-up dialog boxes and windows such as the "Distance from…" windows. This shows the colors used for a list of text elements displayed on those windows. Note that there are two separate columns of color controls shown, one for light mode and the other for dark mode.



To change any of the color settings, click on the color you wish to change. This will bring up a color chooser dialog. Select the desired color and click OK.

If you click the Reset to Default button for a color setting, the light and dark mode colors for that line will return to their default settings.

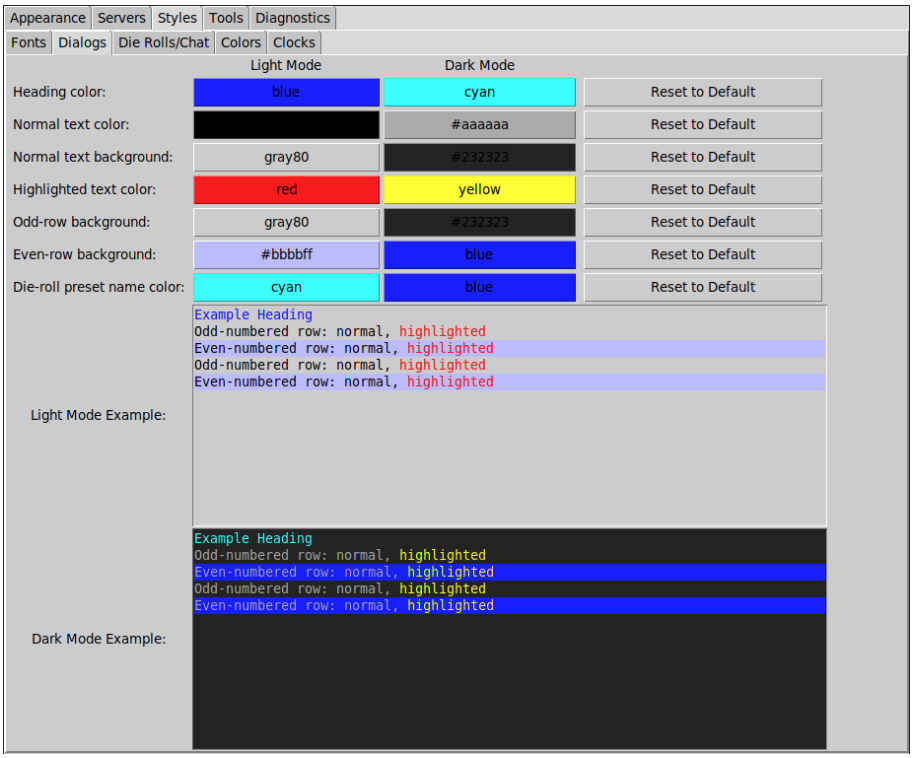


Figure I.4: Dialogs Tab

Styles: Die Rolls/Chat Tab

The "Die Rolls/Chat" tab (shown in Figure I.5) controls the styling of messages displayed in the chat window, including the results of die rolls you or other players make. At the top left, you will see a list of the different styles used by the mapper to display these messages. Clicking on one will select it as the style type to edit. In the above example, we have selected the "roll" style, which is used to show the individual die-roll results (e.g., for a die roll specification such as "3d6", 3 six-sided dice will be rolled and this style used to display the natural value that came up on each of those dice).

With a style selected, you can see and modify all of the attribues that comprise that style. There is also an example showing what the style will look like alongside the other styles. The selected style is underlined in that example display, to make it easier to see which one you are changing.

The attribute controls are:

Text color: Here are a pair of color controls, one for light mode and the other for dark mode. They function as described above for changing

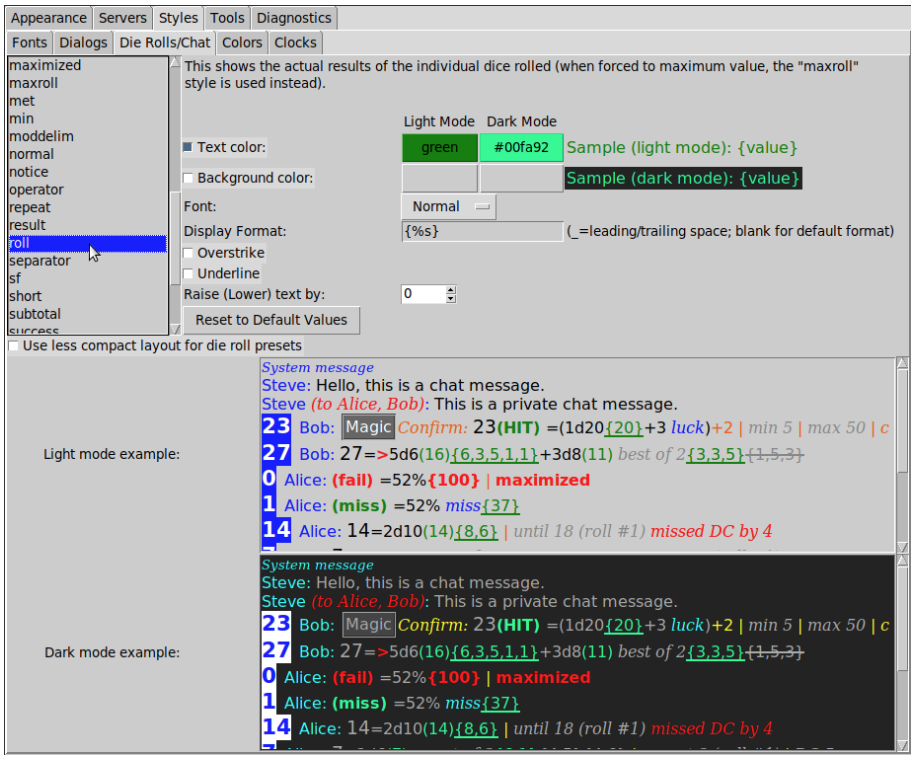


Figure I.5: Die Rolls/Chat Tab

colors in the "Dialogs" tab. This controls the color of text displayed in the selected style.

The color selection will only be valid if the checkbox to the left of this item is checked. If it is unchecked, then no color is specified for this style, and the normal coloring for the window will be used for this style.

Background color: As with the previous set of controls, but these control the background color under the text.

Font: This allows you to choose which of the fonts defined in the "Fonts" tab will be used for text in this style.

Display Format: This controls how values of this style will be written. If this is left empty, the text is written as-is, without any formatting. Otherwise, this is a `printf`-style format string used for the display of the value. This allows you to add words or punctuation around the value. The value itself will be written wherever you have a "%s" in the format. If you don't include a "%s" at all, then the value intended to

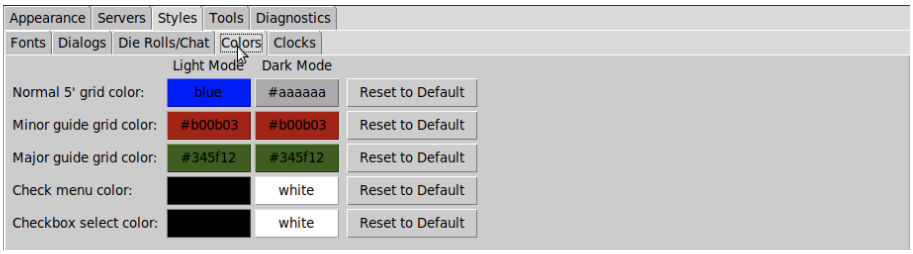


Figure I.6: Colors Tab

go in that part of the message will not appear at all—only the other text in your format will be shown.

A leading or trailing "_" will be converted to a single space. This allows you to add leading or trailing space to the value to be printed. Otherwise, any leading and trailing spaces are ignored. An underscore anywhere else in the format string is simply a literal underscore character.

Overstrike: If this is checked, text in this style will have a line struck through it.

Underline: If this is checked, text in this style will be underlined.

Raise (Lower) text by: If this is non-zero, text in this style will appear the indicated distance above the baseline (or below it, if the value is negative).

Reset to Default Values Pressing this button restores this style to the original default settings.

Use less compact layout for die roll presets If checked, the die rolls shown in the lower panes of the chat window will take up multiple lines if necessary to fit them into a narrow window. Otherwise they will be displayed strictly one per line.

Styles: Colors Tab

The "Colors" tab (Figure I.6) provides access to the colors of the general user interface elements of the mapper, such as the map's grid lines. Changing the colors here works the same as described above for the "Dialogs" tab.

Styles: Clocks Tab

The "Clocks" tab (Figure I.7) controls the styling of the game clock and initiative list display window. The following colors may be set in the same fashion as already described above:



Figure I.7: Clocks Tab

Clock hands: The color used to draw the hands on the game clock, and the circle around it.

Clock tick marks: The color used to draw the tick marks around the clock face.

Initiative list text color: The color to use for the combatants' names in the initiative list.

Initiative list background: The background to use behind combatants' names in the initiative list.

Next round marker color: The text color for the "next round" marker in the initiative list.

Next round marker background: The background color for the "next round" marker.

Current actor background: The background color to use behind the name of the creature whose turn it is.

Readied action background: The background color for the names of creatures with readied actions.

Held action background: The background color for the names of creatures who are holding their action.

Frame: zero hit points: The frame color drawn around the names of creatures with zero hit points.

Frame: negative hit points: The frame color drawn around the names of creatures with negative hit points.

Frame: flat-footed points: The frame color drawn around the names of creatures who are flat-footed.

Negative hit point text color: The text color for the names of creatures with negative hit points.

Negative hit point background: The background color behind the names of creatures with negative hit points.

There are also three fonts you can set here, by picking the name of one of the user-defined fonts on the "Fonts" tab:

Time display font: Used for the text display of the current time and date.

Turn display font: Used for the text display of the round counter during combat.

Initiative list font: Used for the combatant names in the initiative list.

Tools Tab

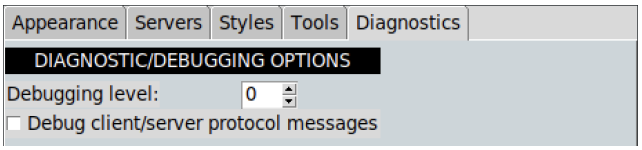


The "Tools" tab allows you to specify the path to helper programs the mapper needs for its operation. The mapper will make an attempt to default this to a path it believes will work, but you can adjust as needed or desired.

Curl program path This is the location on your local system where the `curl` utility may be found. The mapper will use this program to download images and updated copies of the mapper itself from your group's server. (Same as the `--curl-path` CLI option.)

Run Curl in insecure mode If this is checked, the `curl` program will be run with the `-k` option when it is invoked to retrieve images and map files from the server. This skips the SSL certificate verification step that helps ensure you are loading files from the genuine server instead of an imposter. Normally, you do *not* want to check this box, so the security checks will be included. Don't select this unless you understand the security implications of doing so and are willing to accept the consequences.

Diagnostics Tab



The "Diagnostics" tab enables the output of extra diagnostic output that may assist with tracking down a problem in your mapper client or server.

Debugging level The higher this number is set to, the more output will be produced at ever-increasing levels of verbosity. 0 will only display things that are urgent for your information, while 6 provides more information than you probably ever need to see. (Same as the `--debug` CLI option.)

Debug client/server protocol messages If checked, this shows the traffic between the client and server, and how the mapper client interprets incoming messages. (Same as the `--debug-protocol` CLI option.)

Servers Tab

Now let's examine the "Servers" tab (Figure I.8). By default, the mapper will give you a default server profile called "offline" as shown here. At the top left of the panel is a list of all the server profiles you have set up. Whichever one of those is selected (as "offline" is here), that will be the server the mapper client will use every time it's started. In addition, all of the other fields on this panel will be associated with that server profile. The appropriate set of values must be configured for each server profile, since they may change from game to game and server to server.

To the right of the server profile list are three buttons:

Add New... You will be prompted for the name of a new server profile to create. That will become the selected profile and will contain blank or default field values. You should then fill in the remaining applicable fields for your new server profile.

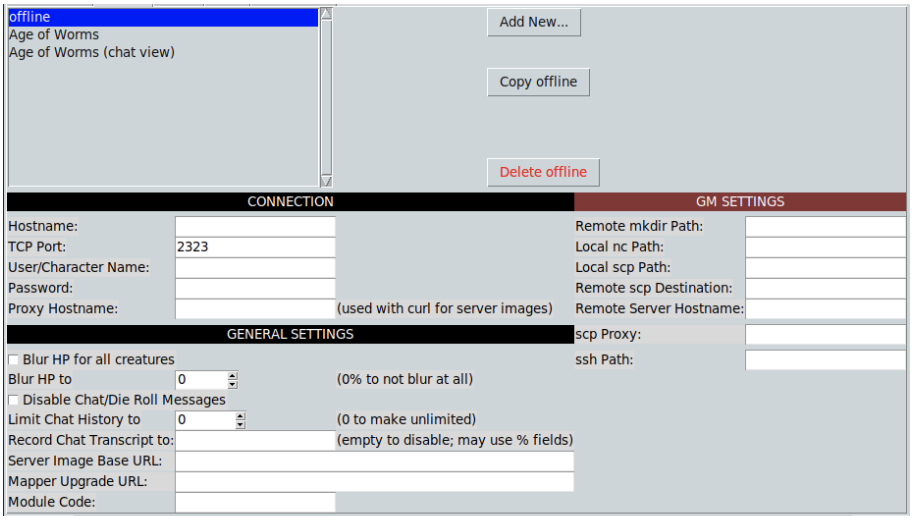


Figure I.8: Servers Tab

Copy ⟨***profile name***⟩ Functions the same as the Add New... button, but will copy all the configuration field values from the currently-selected profile into the new one you're creating. This may make it easier to make a profile that is similar to another existing one, since now you merely have to adjust the values that differ between the two.

Detele ⟨***profile name***⟩ Deletes the currently selected profile from the system.

Say you want to create a new profile called "Example." Start by clicking the Add New... button, then filling in the fields with a set of values appropriate for that server (Figure I.9).

Connection

Hostname The name of the server your GM set up for your game's map server. (Same as the `--host` CLI option.)

TCP Port The TCP port number for your game's server. (Same as the `--port` CLI option.)

User/Character Name This is the name you will be known by on the server. It will appear in the chat window and die-roll results. (Same as the `--username` CLI option.)

Password The password your GM provided to you to log in to the server. *Please note: while this password is not transmitted over the network,*

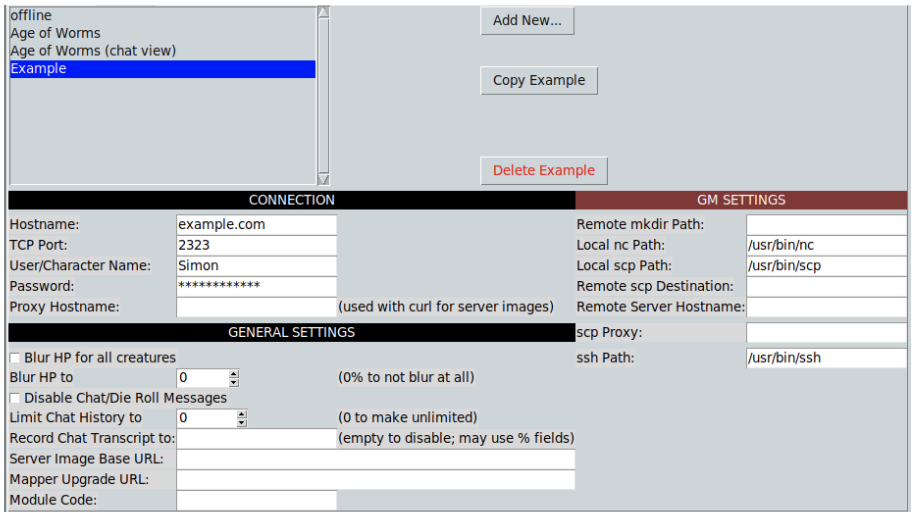


Figure I.9: Servers Tab

it is stored in your local preferences and configuration files ***unencrypted****. Do not use the same password for the game as you use for anything else in your life.* (Same as the `--password` CLI option.)

Proxy Hostname If you connect via a SOCKS5 proxy server, specify it here (e.g., "`proxy.example.com:8080`"). (Same as the `--proxy-host` CLI option.)

General Settings

Blur HP for all creatures If checked, then your setting for "HP blurring" applies to player characters as well as other creatures. (Same as the `--blur-all` CLI option.)

Blur HP to If nonzero, hit points shown for creature tokens will not be precise; instead they will be only shown to the specified percentage. For example, entering "33" means the health bar will only show that the hit point values are 0%, 1–33%, 34–66%, 67–99%, or 100%. Note that even if you specify 0 here, the GM may have configured his or her software to blur hit point values independently, so your mapper won't have any precise values in the first place. (Same as the `--blur-hp` CLI option.)

Disable Chat/Roll Messages If checked, the chat and die-rolling facility is entirely disabled for this client. (Same as the `--no-chat` CLI option.)

Limit Chat History to If nonzero, the mapper will keep a cached copy of only the most recent *n* chat and die-roll messages, where *n* is the number entered here. (Same as the `--chat-history` CLI option.)

Record Chat History to The file pathname where a transcript of all received chat and die-roll messages will be written. If this is left blank, no such transcript will be made. This is not the same as the message cache the mapper keeps internally. This pathname may contain special values such as `%m` for the month number and `%d` for the date. For the full list of such values, see page 477. (Same as the `--transcript` CLI option.)

Server Image Base URL The URL on your game server where map image files may be found. This is supplied by your GM. (Same as the `--curl-url-base` CLI option.)

Mapper Upgrade URL If your GM has set up part of their server to host signed copies of the mapper client, they will provide you the URL value to enable automatic upgrades from within the mapper program. If they want you to upgrade your client from the public repository but wish to recommend what version you use, set this to the public distribution URL:

"`https://raw.githubusercontent.com/MadScienceZone/gma-mapper/main/signed-releases`".

(Same as the `--update-url` CLI option.) In any event, you can always manually upgrade to the latest public release by choosing Tools 〉 Check for Updates... from the main menu.

Module Code The ID code for the adventure you're playing. (Same as the `--module` CLI option.)

GM Settings

As the name implies, these values are only needed if you are the GM sending content *to* the server for your players to access.

Remote mkdir Path The pathname to the `mkdir` program on the remote server. This will be used to create new directories server-side as needed. (Same as the `--mkdir-path` CLI option.)

Local nc Path The pathname to the `nc` program on your local system. (Same as the `--nc-path` CLI option.)

Local scp Path The pathname to the `scp` program on your local system. (Same as the `--scp-path` CLI option.)

Remote scp Destination The pathname *on the remote server* of the top-level directory where image and map content will be located for players to access. This is usually a directory served by your web server, so that the `curl` program run by everyone's mapper can find them. (Same as the `--scp-dest` CLI option.)

Remote Server Hostname The name of the game server as given to programs such as `ssh` and `scp`. You may prefix the hostname with "⟨*username*⟩`@`" if your username on the remote server is not the same as on your local system. (Same as the `--scp-server` CLI option.)

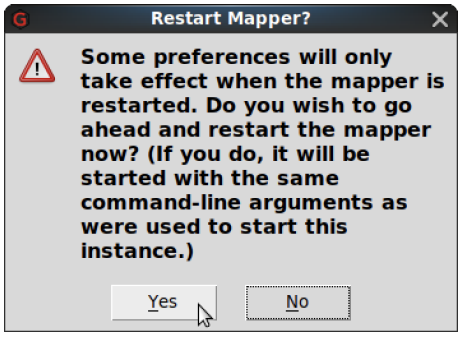
scp Proxy (Same as the `--proxy-host` CLI option.)

ssh Path The pathname to the `ssh` program on your local system. (Same as the `--ssh-path` CLI option.)

Saving Your Preferences

When you click the Save button at the bottom of the preferences window, those settings will be saved for the mapper to keep using. However, at this point it may already be too late for some of the new settings to take effect, since the mapper has already initialized itself from what it knew at start-up time.

Therefore, the mapper will ask you if you want to restart the program, so it starts off with these newly-set preferences values in effect:



Command-Line Options

If you specify explicit command-line options to set any of these values, they will override the settings loaded in from your saved preferences at program start-up. If you then edit the settings manually as described above, those new settings will take precedence unless you choose to restart the mapper after saving the new preferences.

Temporarily Selecting a Server

You can use the `--select` CLI option to name a server profile to use at program start-up. This will select that as the profile to use, but just this one time—it does not change the default server profile name all subsequent clients will use, as selecting a profile name in the preferences editor does. Note that if the mapper was started with the `--select` option, it will keep obeying that option if it's internally relaunched by answering Yes to the preferences editor's final dialog as described above, or the File › Restart Mapper menu option.

You can start the mapper with the `--list-profiles` option to list all the available profiles you can name with the `--select` option.

If you can't use the command-line, the recommended approach to switch servers is to select the desired profile in the preferences editor, save, and let the mapper restart.

Legacy Configuration Method

In mapper versions prior to 4.4, configuration of the mapper was done entirely via command-line options and a simple configuration file located at `~/.gma/mapper/mapper.conf` which uses the same option names as the CLI options (omitting the leading "`--`"), separating the option from its value with an equals sign ("=").

This is no longer recommended and the `mapper.conf` files are deprecated in favor of the in-app Edit › Preferences... menu.

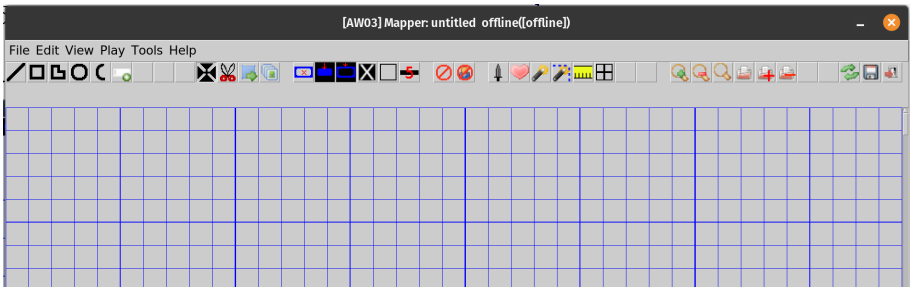
Starting the Mapper

Now make sure your mapper is installed and working correctly by starting it up. On Windows you can open the `$GMA_MAPPER` folder (whatever you actually called it), then open the `bin` subfolder and finally double click on the `mapper.tcl` file. On *nix systems with the full GMA Core suite installed, running `gma mapper` will work. Otherwise, just run the `bin/mapper.tcl` program directly. In other words, here are a few examples of how to start the mapper, depending on how your system is configured and how you installed the mapper.

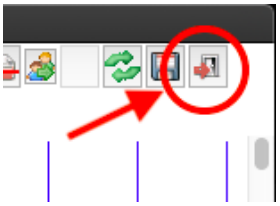
```
$ gma mapper
$ $GMA_MAPPER/bin/mapper.tcl
$ wish $GMA_MAPPER/bin/mapper.tcl
$ wish8.6 $GMA_MAPPER/bin/mapper.tcl
$ wish8.7 $GMA_MAPPER/bin/mapper.tcl
```

(The above assumes the `gma` command is in your `$PATH` variable and that `$GMA_MAPPER` is the location of the `gma-mapper` top-level directory as you installed it.)

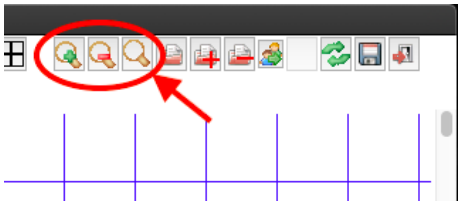
If all goes well, you'll see the mapper window with a blank map and blue grid lines. (If the mapper detected your system is using a dark color theme[1] or if you checked the dark mode preference, the mapper will adopt its own dark theme, showing a black map background with light brown grid lines.)



Most of the buttons on the tool bar are things you won't need to (and probably shouldn't) click on during an actual game, but you'll want to at least know about the exit button on the right, which will terminate the mapper program:



. . . and the zoom controls which will let you adjust your view of the map:



The zoom-in button () zooms in so everything you see is twice as large as before, while the zoom-out button () does the opposite. Regardless of current zoom level, the unzoom button () resets to the normal view.

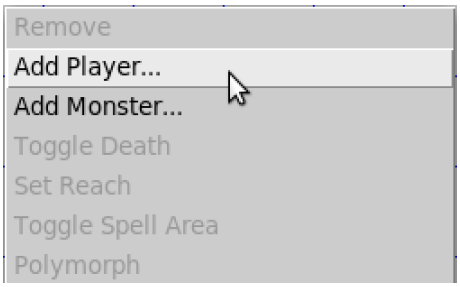
On the far left are a number of buttons to select drawing tools used when modifying the map itself. You won't usually be doing that as a player, but the one button you should know about is this one:

[1] Not possible for all operating system platforms.



This selects the normal play mode, which is the default one that's active when the mapper starts. This is the tool you want active while you play the game. With this, you can pick up and move around your character token. Since we're not actually playing at the moment, let's add a token to practice with.

Right click on the battle map area (where the blue grids are). Select Add Player... from the menu that pops up.

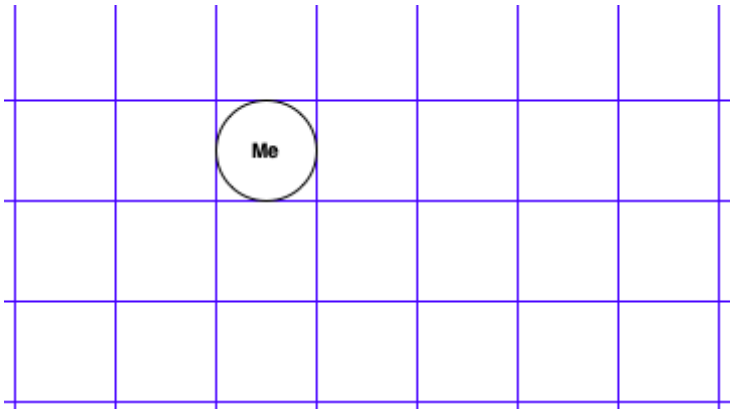


This will bring up a dialog box where you can enter the name of your token and other attributes such as creature size.[2] Type a name and click Ok.

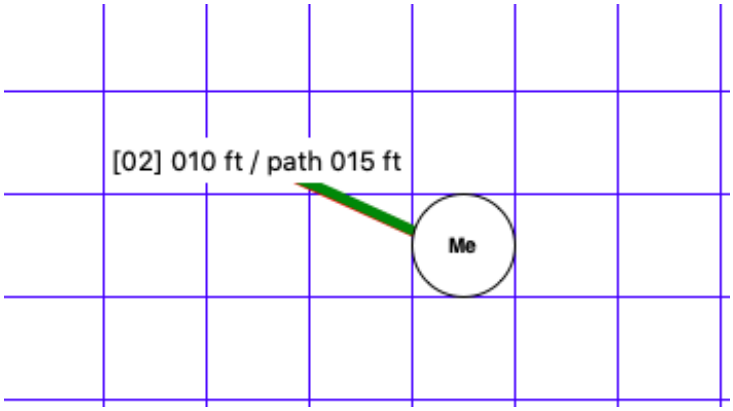


Now you have a token that represents you on the map.

[2]For creatures of different sizes, the "size categories" field can be given any of the forms of size specification as documented in the character sheet file format on p. 410 (`gma-character-record`(5) entry, `[animal.`⟨*id*⟩`]` stanza, `size` field). For example, "`M`," "`L`," "`M->15`," or "`S0`." If multiple size specifiers are entered, separated by spaces, then that creature will have that many "skins" (alternative images) defined for it, which can be selected by the "Polymorph" context menu option.



If you click and drag the mouse on your token, you can move it around the map. Note that while you move it, the mapper will show you the path you dragged it around, and the distance along that path as well as the straight-line distance between your starting and ending points.



If you click and drag the mouse on the map someplace other than over a creature token, that will grab the map itself and drag it, allowing you to pan the view in any direction.

You can also go to a specific map location if you know the grid coordinates (such as "X45") by selecting View > Go to map location... from the menu.

Play around with the mapper as much as you like, and click the "Exit" button ( ) when finished.

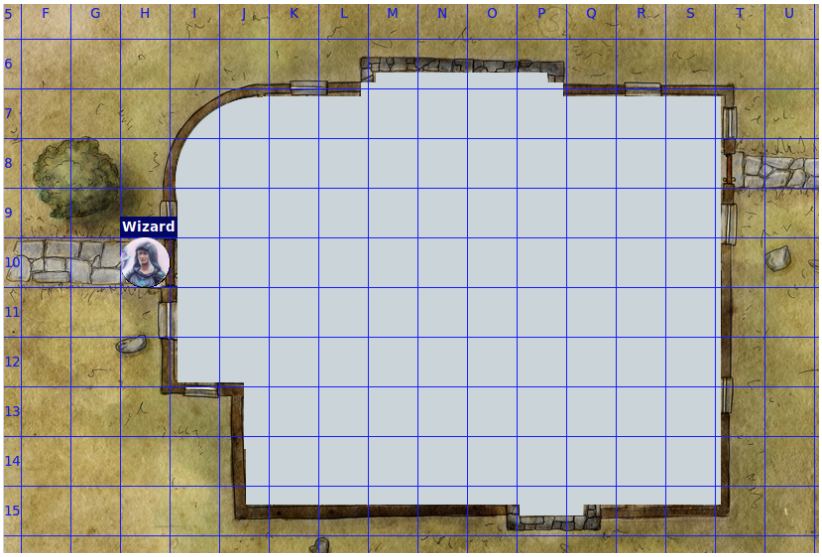
Let's Play!

For an actual game, your mapper will need to connect to the game server we use so you see the environment around you and where everyone else is. To enable this, you need to add a server profile in your preference settings as described above.

Now when you start your mapper, it will connect to the game and will receive updates as the GM and other players move things around on it. Everyone else will also see changes you make to the map. Conceptually, the group is sharing one map; everyone's individual mapper programs let them view and update the shared map.

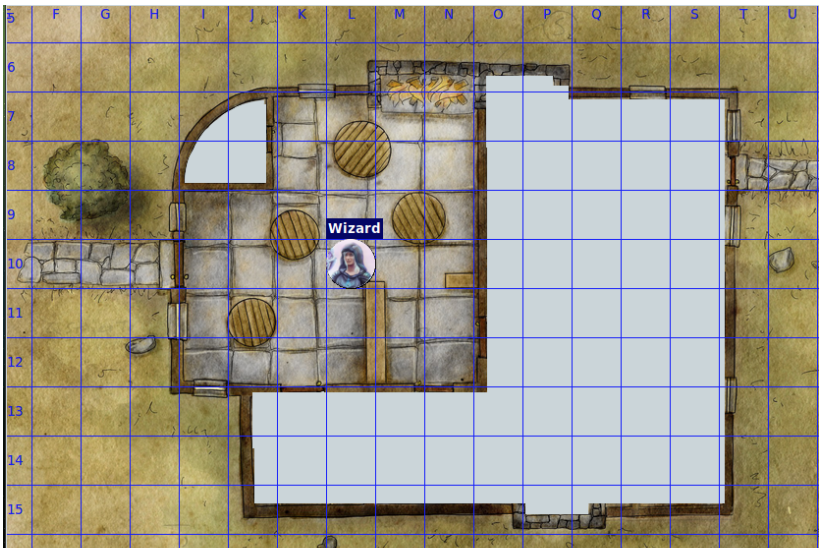
When your mapper connects to the game server, it will receive game-specific information such as pre-made tokens for the party members. Your toolbar will also go away to give you a little more screen space for the map and to help avoid mishaps by accidentally pushing buttons that would erase or modify the map for everyone. As a player, though, most of what you need is available by dragging creature tokens as we practiced above, and right-clicking to get the pop-up menu. The toolbar can be turned back on as needed, as well.

As you explore, the GM will add details to the map, giving you a view like this:



Art Credit: Jonathan Roberts

As you can see, only the parts of the area you've seen yet are displayed on the map. Here, a wandering wizard has approached an inn, but since she hasn't stepped inside she can't see the interior yet. When she opens the door and enters the building, the GM adds the main room to the map.



Oh, no! A fearsome fire monster appears out of the fireplace![3]



When we go into initiative, the mapper will display tactical information including whose turn it is and what space around them they threaten.

[3]Art credit: Deevad



In this view, you can see the threat zone indicated as a dashed circle around each person. Typically, the zones around PCs are drawn in blue, allies in green, and adversaries in red.

When it's someone's turn to act, the specific squares they threaten are shaded in, like this:



If the wizard moves forward one square, we see that she is now in range to attack.



Health Bars

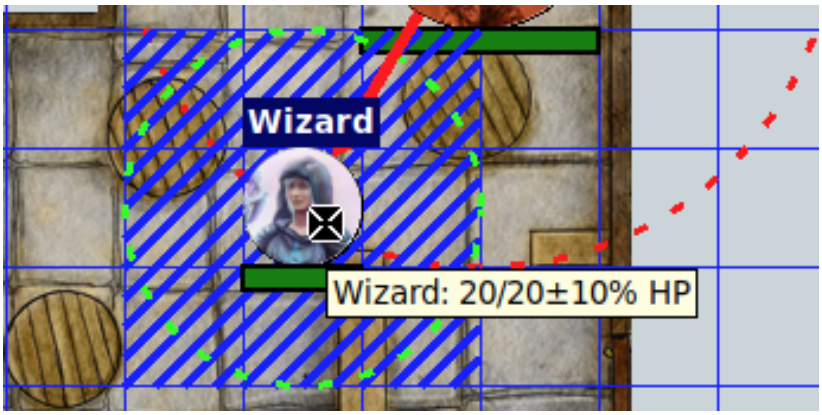
While in combat, a health bar appears under each creature token to indicate how injured they are. The border around the health bar indicates status (flat footed, unconscious, dying, etc.) while the colors in the bar show level of injury. Here are a few examples of what these may look like: (see Figure I.10)

Your GM may configure the system so that these values are only approximations of everyone's health rather than an accurate view.

If you hover your mouse over a token, their name and health status will be shown in a pop-up window. (If the GM has made health bars approximate, these numbers will be, too, either for your opponents or for everyone.)



Figure I.10: Example Health Bars

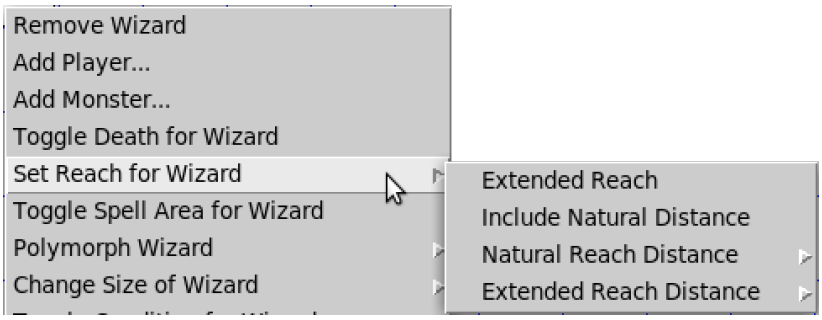


Indicating Special Conditions and Movements

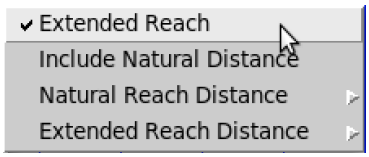
Sometimes you need to indicate special situations with your character on the map. Here are a few examples and how to do them.

Reach Weapons

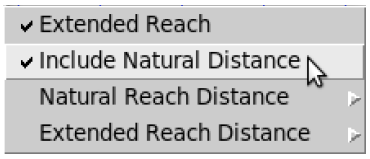
If you wield a reach weapon, you just need to right-click on your character and select Set Reach from the menu.



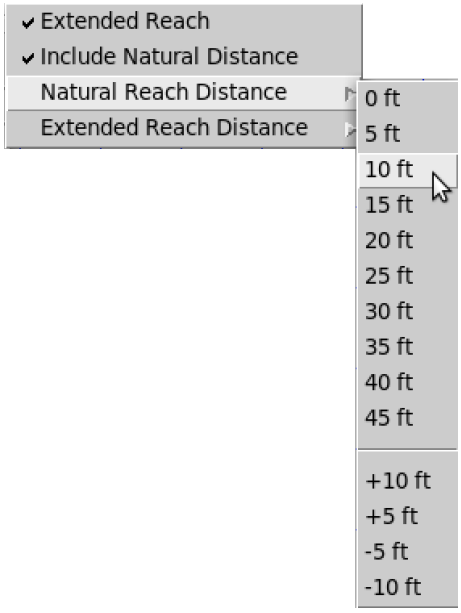
This brings up four options.



Clicking on Extended Reach will indicate a wider threat zone for you when in combat, but remove the squares adjacent to you, since you can't reach them with a long weapon in most cases. Selecting this item again turns it off, returning to the normal threat zone area for your character.



Clicking on Include Natural Distance while Extended Reach is selected will include the full area, including the squares adjacent to you. This is used for special attack forms which allow you the extended reach without losing access to the nearby squares too.



If you need to change your character's natural reach distance, you can select Natural Reach Distance and then either select the specific distance you can reach, or choose to adjust its current size by ± 5 or ± 10 feet.

Choosing Extended Reach Distance does the same, but adjusts the limits of your extended reach (i.e., with a reach weapon) instead.

Area of Effect Spells

If you cast a spell that creates an effect centered on you, right-click on your token and select Toggle Spell Area.



Move your mouse away from your token until you have highlighted the correct number of squares around your token to indicate the spell's effect zone, then click the left button. This zone will be drawn using your mapper's current fill color (you can change this by clicking on the fill color chooser button on the toolbar). If we select a 10-foot area of effect around the wizard, we will see a result as shown in Figure I.11.

This will move along with your token. Right-click and select the same menu option again to turn it off.

Flying, Swimming, and Other Forms of Movement

It is normally assumed that your character is walking. If you're doing something else, right click on your token and select Set Movement Mode and then the actual mode from the sub-menu. Figure I.12 shows how we change the wizard's mode of transportation to flying.

If you are not standing firmly on the ground, you should also indicate your elevation level by right-clicking and selecting Set Elevation and then the amount you're changing the elevation by.

This will display the elevation on your token in a color that indicates the movement mode.



Figure I.11: 10-foot spell area of effect.

Standard Conditions

The game includes a number of standard conditions a creature may be in. Some of those, such as *dying*, *disabled*, and so forth, we have discussed above when showing how the health bars work. However, the full set includes states which don't necessarily indicate "health" status or need to be shown when not in initiative (i.e., when the health bar isn't shown).

To set one of these conditions, right-click on a creature token, select Toggle Condition and then the specific condition (see Figure I.13).

Now when you hover your mouse over the creature token, the health information will include an explanation of the condition(s) in effect for that creature.

The selected condition is *toggled*—if the creature did not have that condition, it is set on them; if they did, it is removed. To add multiple con-

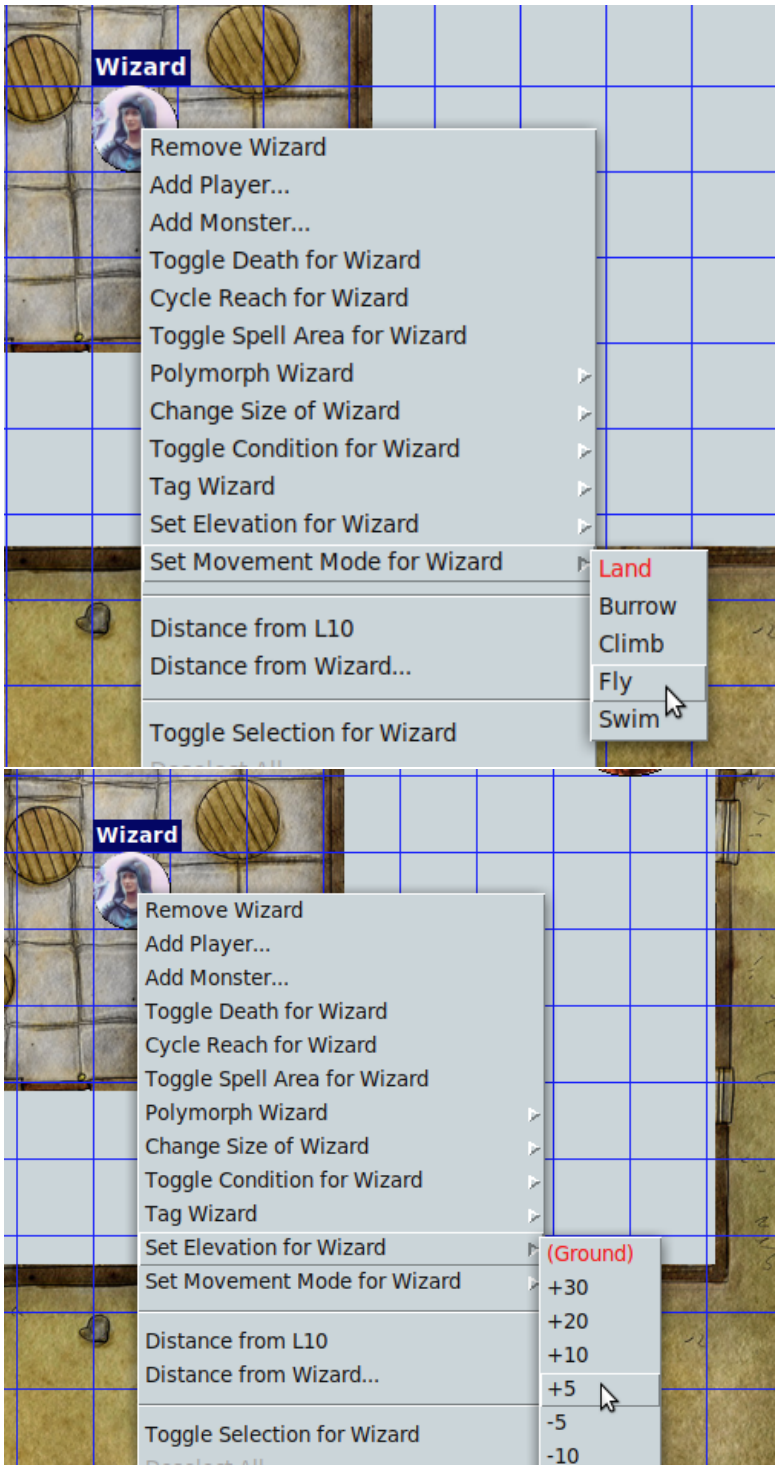


Figure I.12: Indicating that a creature is flying at height of 5 feet.

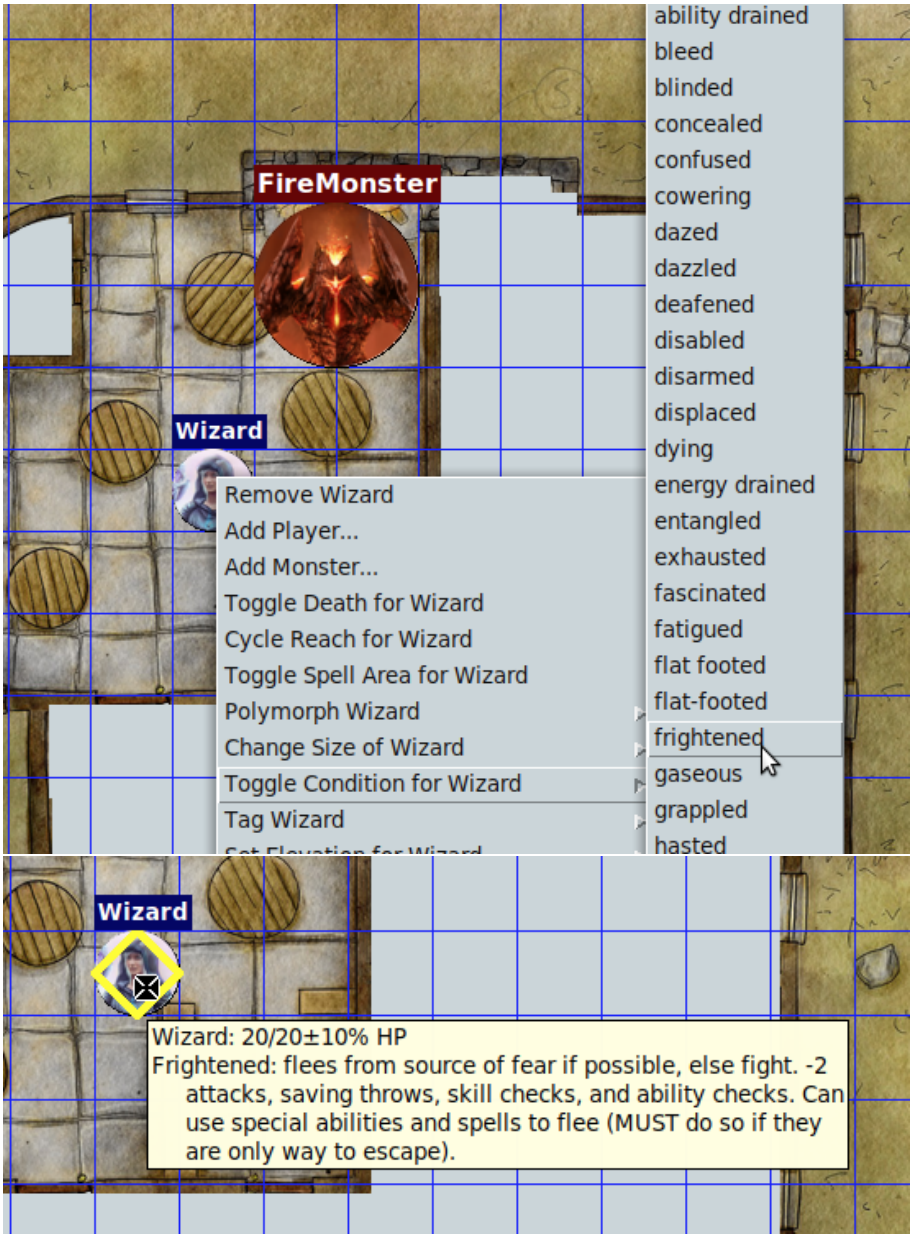


Figure I.13: Setting a condition on a creature.



Figure I.14: Example Condition Markers

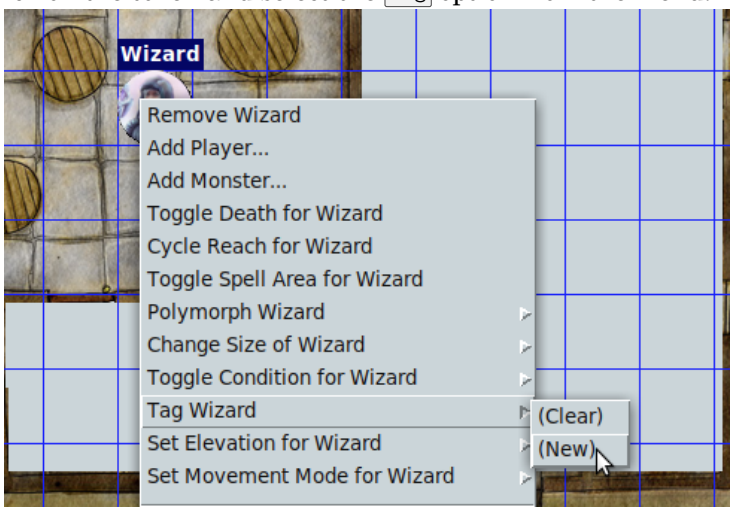
ditions, just repeat this operation for each condition. If the special item Toggle Condition > (clear all) is selected, all conditions are removed from the creature. Note that merely clearing a condition like "dying" manually won't really remove this status if the creature's hit point total indicates that they are, in fact, dying.

The full set of conditions available, and the way each is marked on the map, is completely configurable by the GM (server-side), but these are the standard example markers the mapper tool comes with by default: (see Figure I.14)

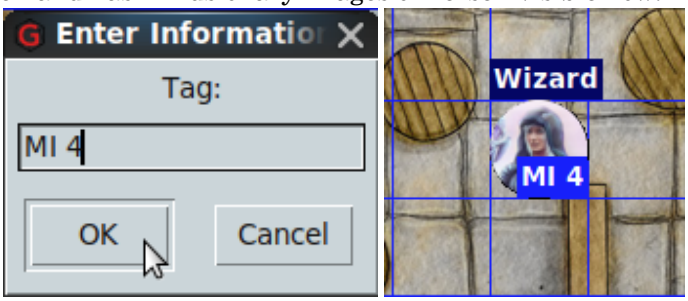
The intent is for the colors to be at least a little suggestive of the condition, or memorable, but there is by necessity a fair bit of arbitrary assignment of colors and symbols. Generally speaking, conditions which may be longer-lasting or conditions a character may need to just deal with while continuing with the adventure for some time are shown as small "badges" at the left or right side of their token while more "severe" conditions (in the sense of their impact on short-term actions or combat) are shown as lines around or through the token. Similar symbols are given for "similar" conditions (e.g., a double horizontal line through the token for several conditions which limit the actions a character may take, double diagonal slash for visual impairments, etc.).

Special Conditions

If there's anything else that should be noted about your character, that isn't covered by the condition markers, such as having mirror images in effect, right-click on the token and select the Tag option from the menu.



Select Tag 〉 (new) to create a new tag or pick an existing one if what you want is already listed there. If you make a new one, you will be asked to type in the tag text. Let's indicate that the wizard has cast the *mirror image* spell and has 4 illusionary images of herself visible now:



When the tag no longer applies, right-click and select Tag » (clear).

Gridlines

By default, the mapper displays gridlines at 5-foot intervals. If this interferes too much with the appearance of the map, they may be turned off by clicking the grid button on the toolbar (⊞). Be sure not to confuse it with the grid-snap button. They look very similar.

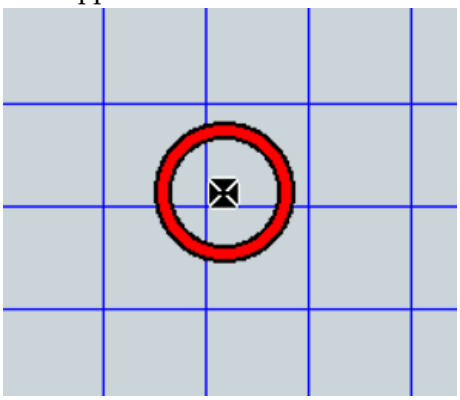


See Figure I.15 for a comparison of what the map looks like with and without grid lines.

Pointing to Areas of Interest

If you want to draw other players' attention to a particular spot on the map, hold the shift (⇧) key while you click the left button on the spot in question.

This will cause a brief animated circle to display at that location on everyone's connected mapper clients.



Measuring Distances

The ruler tool is activated by clicking on the ruler icon (📏). Click and release the left button on any location on the map, then move the mouse to another location. The ruler tool will display the distance between those two points, in even grid units as well as in feet. Note that the measurement is always in even grid units—the feet are simply a translation from grids to feet.

You can measure the distance along a path by clicking the left button again at the point where you want to change direction. Figure I.16 shows an example of this.



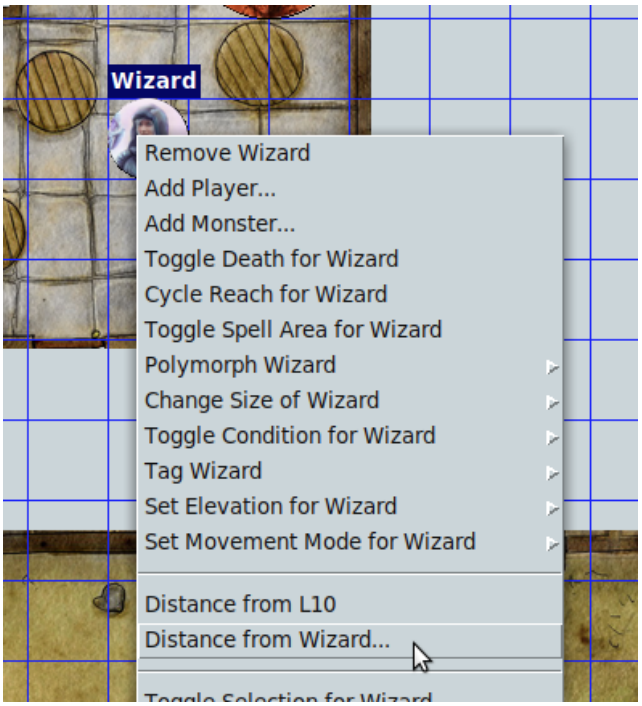
Figure I.15: Mapper With and Without Gridlines.



Figure I.16: Using the ruler tool to measure along a path

Distance to All Creatures

Sometimes you need to know which allies or enemies are within range of a spell or other area effect. There are two ways you can ask the mapper to calculate the distance to each creature on the map in 3-D space.



If you right-click on a creature token such as the Wizard in our example, and select Distance from Wizard..., this will bring up a window listing every other creature on the map and their distance away from the Wizard.

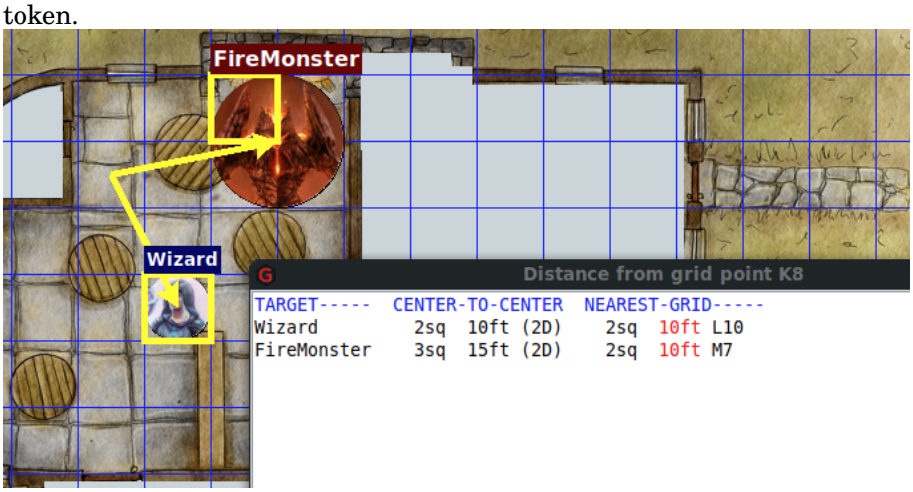


In this case, the Fire Monster is 3 grid squares away, or 15 feet, if we measure from the center of the Wizard's creature token to the center of the Fire Monster's. However, it is usually more useful for game mechanics to know the distance between the nearest squares occupied by each creature. That distance is shown on the right side of the displayed distance table under "nearest-grid." Here we see that the nearest-grid distance between them is only 2 grid squares, or 10 feet. It also says that the specific square on the map that is closest to the Wizard is grid number M8.

If creatures are at different elevations, the mapper will accurately account for the distance in 3-D space and report that. This shows the actual distance in three dimensions, not based on a game mechanic rule intended to simplify 3-D distance calculation. Since we have a computer doing the heavy lifting for us, we might as well use it to get accurate measurements.

As with the ruler tool, remember that the measurements are shown in even grid units with the number of feet shown for convenience.

You can also measure the distance from any arbitrary map square by right-clicking on that square (say, grid K8), select Distance from K8, and get the same window showing the distance from that square to every creature token.



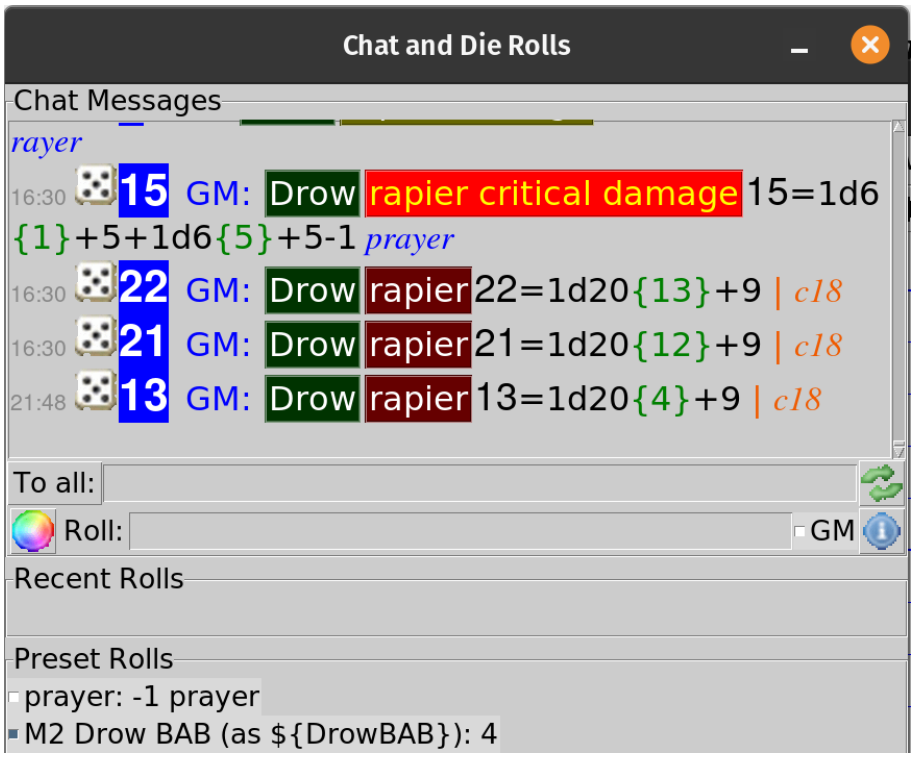


Figure I.17: Chat and Die Rolls Window

Note that sometimes the square highlighted by the mapper as being "closest" isn't the one that is obviously the nearest in direct line of sight between the creatures. This happens if there are multiple squares that are the same distance (in even grid units) away, and all of them are the minimum distance. While the exact square it chose to highlight may not be the correct one, the distance measurement is correct.

Chat Messages and Die Rolls

Clicking on the die icon ( ) opens the Chat and Dice Rolls window as shown in Figure I.17. Full details on how you can use this window to make die rolls online and send messages to other players may be found in Chapter 10 on page 187.

Combat Initiative Clock Display

Clicking on the clock icon ( ) opens the Game Clock window as shown in Figure I.18. During the course of normal play, this will display an analog



Figure I.18: Game Clock During Combat

and digital time display showing the current time and date in the game world.

When you go into initiative (such as at the start of combat), the display shifts to a "stopwatch" view. On the clock face, the longest hand sweeps around once per round (6 seconds), dragging a red shaded region behind it to make it easy to see how far into the round you are. The next hand sweeps around once every 10 rounds (1 minute), dragging a green region behind it. The shortest hand sweeps around once every 10 minutes.

Under the clock is a list of combatants, in the order in which they can act. The name of the person whose turn it is will be highlighted. Special highlight colors also indicate if anyone is holding (delaying) their action or has a readied action prepared.

Timers

During the game, the GM may set up a number of timers behind the screen which mark the time until some event takes place in the game. Some of these, such as when a spell you cast expires, are directly applicable to your character and something your character would be aware of. The GM can

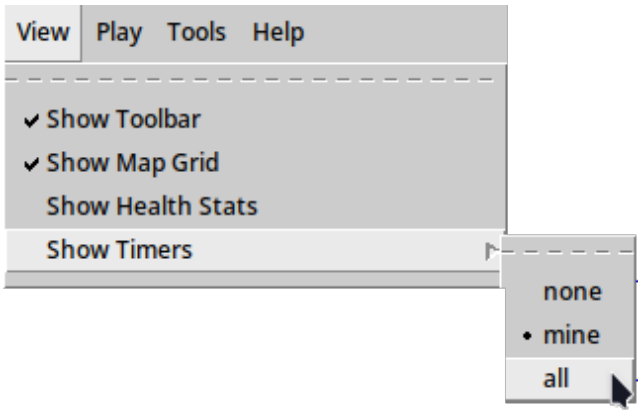


Figure I.19: Game Clock Timer Menu

choose to expose those timers so you can see them on your mapper. If they do so, they will appear on the Game Clock window between the clock itself and the initiative list.

By default, only timers which specifically apply to you will be shown.[4] You can change this by selecting the View > Show Timers > all menu option to display all timers, or View > Show Timers > none if you don't want to see any timers at all. See Fig. I.19

Keeping the Current Actor Visible

If you select the View > Scroll to follow combatants menu choice so that it is checked in the menu, the mapper will auto-scroll so that the person whose turn it is to act in combat. This is useful if the combatants are spread across a wider area than will fit in your map window at once.

Spell Area of Effect

When casting a spell which has a range and area of effect in the shape of a certain radius, or a cone or line of a certain length, you can indicate the affected grid squares on the map. Begin by clicking the spell icon ( ). Then select the spell area's shape. This is selected by clicking the shape button to the left along with the various shape drawing tool buttons. The default is a radius ( ).

[4] This is determined by the GM explicitly indicating your character as a target of the timer's event. This also requires that you are logged in to the mapper with exactly the same name as your character (including capitalization), assuming the GM set the target name to your character name (which is most likely).

Click and release the left button on the location the spell effect starts. Move the mouse to a point on the outside of the area and click again to complete the area of effect indicator. This is shown in Figure I.20.

You can change the color used to mark the affected area by clicking the toolbar button to set the current fill color ( ) before using the spell area of effect tool.

If you click on the spell shape button, it will cycle from radius to cone to line and then back to radius again.

Marking spells of those shapes works the same as with radius areas of effect. See Figure I.21.

To remove a spell area of effect zone from the map, use the delete tool ( ).

Fractional Grid Snap for Creatures

Sometimes the map areas where you find yourself fighting don't line up right along the five-foot grid lines, which makes arranging people in combat awkward as shown in Figure I.22 (top). It seems in order to be in the corridor, Alice and Charlie need to stand half-way into the walls even though the corridor is five feet wide, just because it doesn't run along the grid lines the creature tokens are snapped to.

To fix this, select the Play » Creature token grid snap option from the menu. This lets you choose between four settings:

By creature size (the default) causes creatures of Small or larger size categories to snap to whole grid-lines, and smaller ones to snap to half-grids and quarter-grids as appropriate.

Full squares Snap all creatures to full grid lines.

1/2 squares Snap all creatures to 1/2-grid increments.

1/4 squares Snap all creatures to 1/4-grid increments.

In this case, we select Play » Creature token grid snap » 1/2 squares to get the result shown in Figure I.22 (bottom).



Figure I.20: Marking a radius spell area of effect

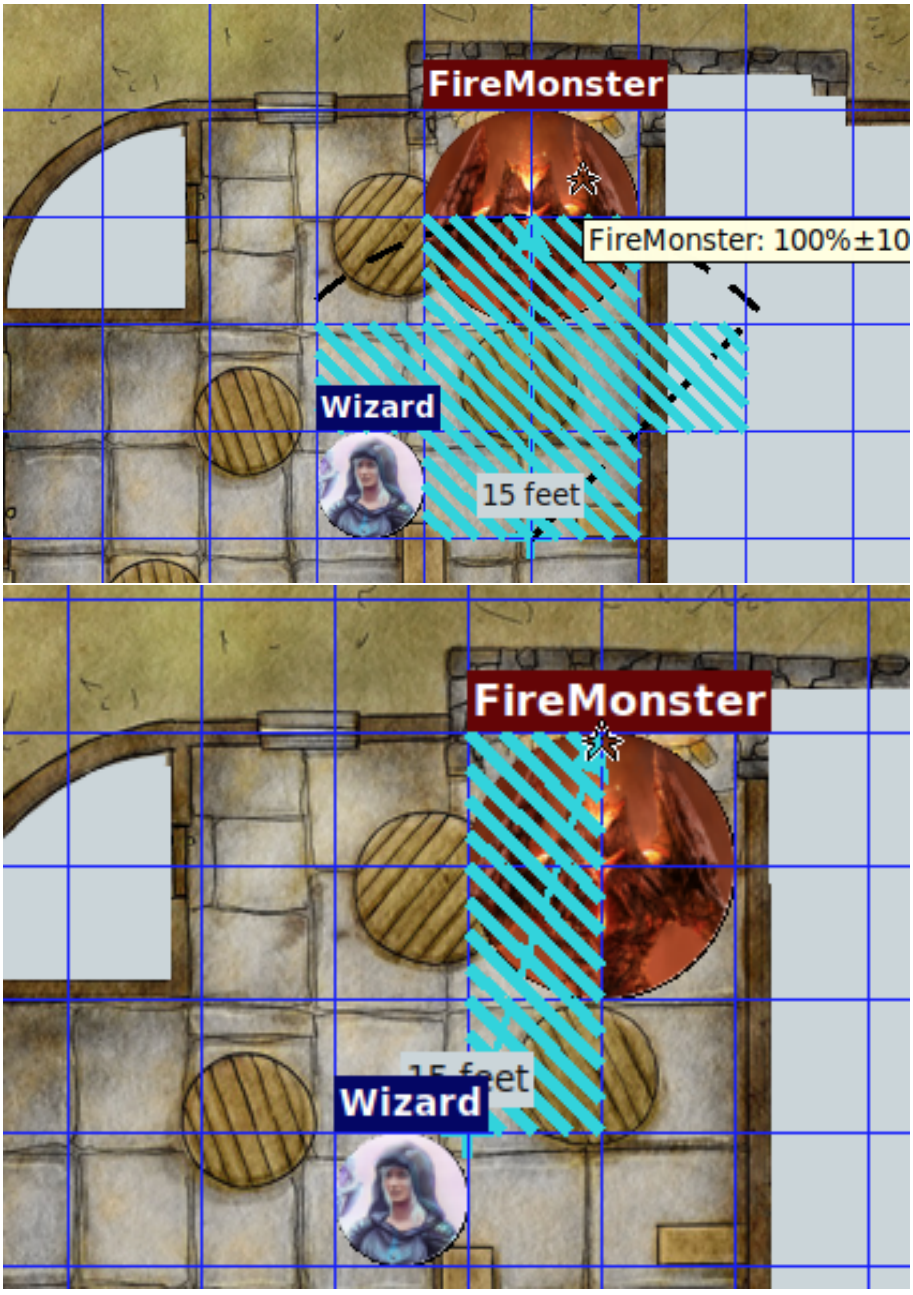


Figure I.21: Marking a cone and line area of effect

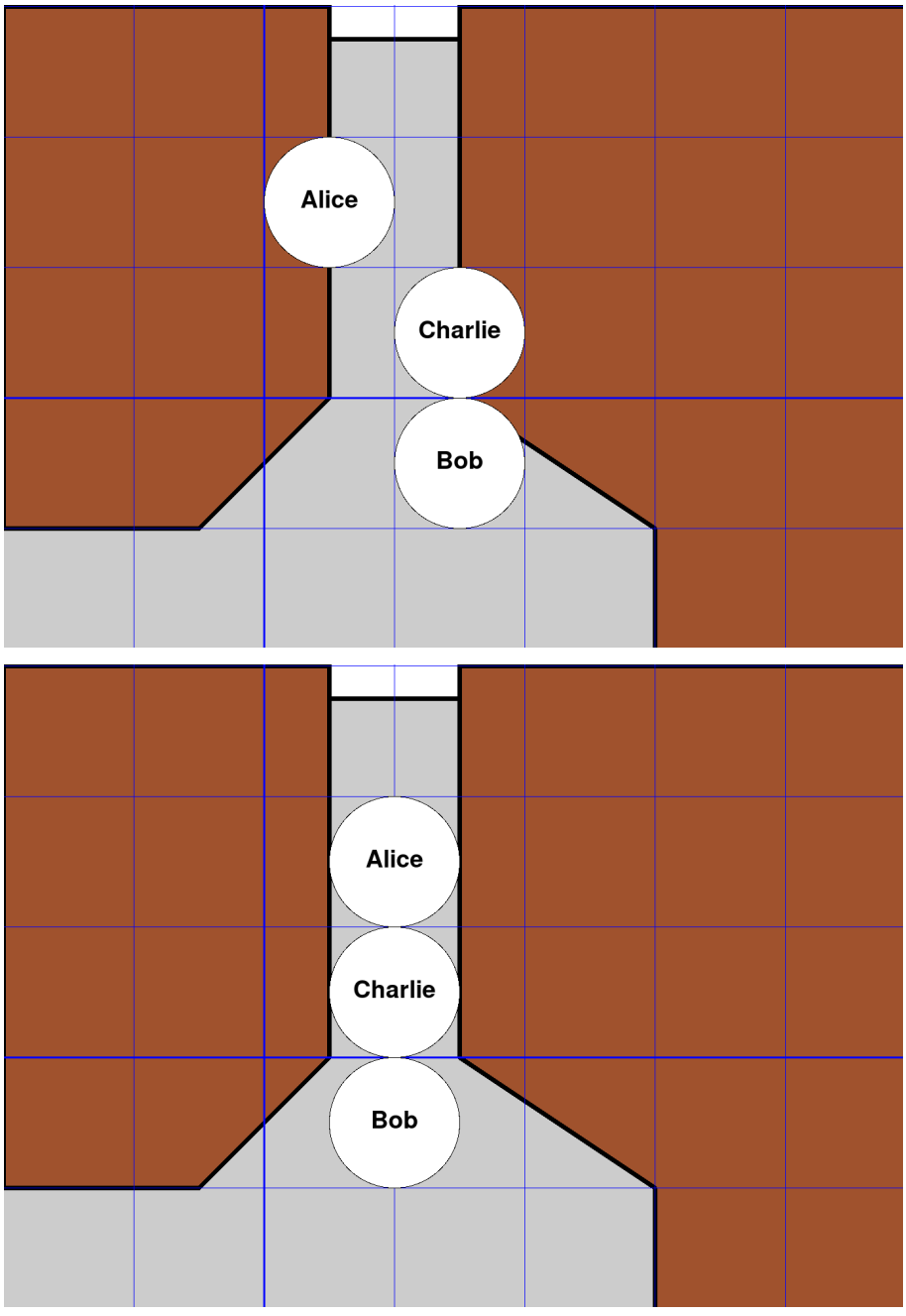
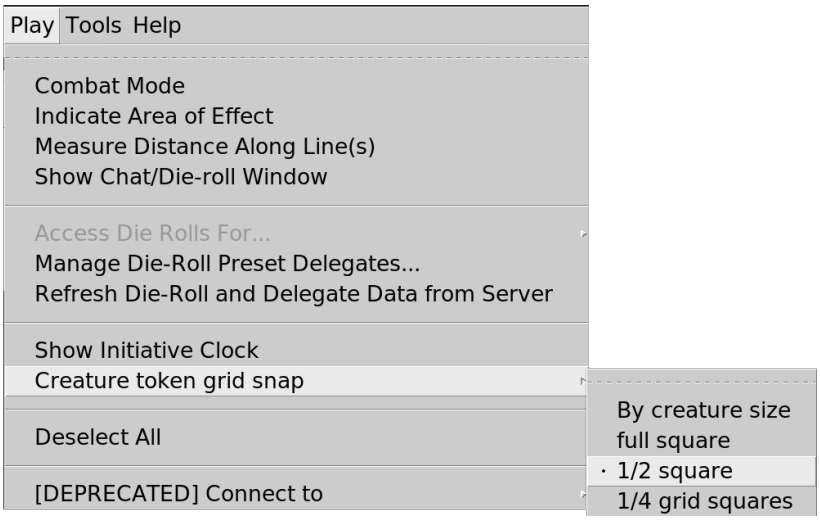


Figure I.22: Creatures Fitting into Misaligned Spaces



It is important to note that the game mechanics are based on the idea of creatures threatening and fighting within 5-foot grid squares. If you shift some creatures to a different grid alignment, you must do the same for *all* creatures they're engaged in combat with.

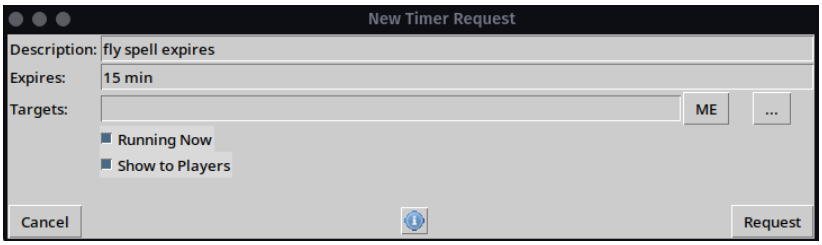
Requesting New Timers

During the course of the game, the GM keeps track of timed events that will either take place at specific dates and times, or which will happen after a specified amount of time has elapsed. For example, if you cast a long-running spell that will last for 15 minutes, they may wish to set a timer to remind them of that expiration time so they know exactly when it will stop functioning. In precision tactical situations such as combat, knowing the exact moment when something like that happens can be important, which is why GMA has a facility to keep track of that for you.

Since a lot of the events (like casting spells) that may result in the creation of timers are caused by player actions, and the GM may be in the middle of doing a lot of things all at once when setting up an encounter, it may be helpful to have each player be in charge of creating their own timers for long-running effects they start. At other times, it may just be the case that you have something random you want to be reminded about later and want to ask for that call-back when the timer expires.

Starting with version 4.28 of the mapper client, you can now set up timers yourself, offloading some of the burden of this chore from the GM.

To initiate a request for a new timer, either click on the toolbar button () or select Play » Request a New Timer from the menu. This will bring up a request dialog for you to fill out:



Fill in the purpose of the timer as you wish it to be displayed on-screen to everyone, and the time when the timer will expire. In this case, our timer indicates that our *fly* spell expires in 15 minutes.

There are two kinds of timers you can request: *absolute* and *relative*.

An absolute timer counts down to a fixed point in time that is significant, like setting an alarm clock to remind you of an appointment, or reminding the GM that at sunset on Thursday the goblins are planning to invade. As they do.

To create an absolute timer, the value you enter for the "Expires" field must begin with an "`@`" character followed by a date and time in the general format

`@`[[[⟨*year*⟩`-`]⟨*month*⟩`-`]⟨*date*⟩] ⟨*hour*⟩`:`⟨*minute*⟩[`:`⟨*second*⟩[`.`⟨*tenths*⟩]]

For example:

- `@12:00` (noon today)
- `@17:30:45` (half-past 5 PM plus 45 seconds)
- `@3-15 8:00` (8 AM on the 15th day of the 3rd month)
- `@Absalom-20 10:15` (10:15 AM on the 20th day of a month called Absalom)
- `@ABS-20 10:15` (same, using abbreviated month name)
- `@4722-GOZ-1 00:00` (midnight, 1st of Gozran, 4722)

On the other hand, a relative timer indicates a length of time that the timer counts down from until it expires, like a stopwatch. To request one of these, the "Expires" field value may begin with an initial "+" or "-" sign to indicate that the timer is to expire after a length of time in the future, or notes the passage of time from an event in the past, respectively. The default is to set the timer for a future event, so usually you can omit either sign entirely. The general forms this value may take are:

[±][⟨*n*⟩] [⟨*units*⟩]
[±][[⟨*days*⟩`:`]⟨*hours*⟩`:`]⟨*minutes*⟩`:`⟨*seconds*⟩[`.`⟨*tenths*⟩]

For example:

- `+5 rounds`

- `10 minutes`
- `1:2:3:4.5` (one day, two hours, three minutes, and 4.5 seconds)
- `10` (10 initiative counts, i.e., 1.0 seconds)
- `round` (1 round, i.e., ⟨*n*⟩ defaults to 1 of the given unit)
- `3 days`
- `1.5 hours`
- `4 weeks`

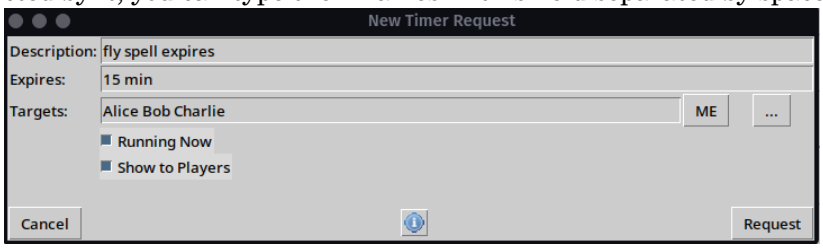
The unit names you can use are seconds (including "seconds," "second," "secs," "sec," and "s"), rounds (including "rounds," "round," "rnds," "rnd," and "r"), minutes (including "minutes," "minute," "mins," "min," and "m"), hours (including "hours," "hour," "hrs," "hr," and "h"), days (including "days," "day," "dys," "dy," and "d"), and weeks (including "weeks," "week," "wks," "wk," and "w").

There are two check boxes on the form, both of which are selected by default, and in most cases should be left that way:

Running Now —if selected, the timer will begin running as soon as the GM accepts it into the system. Otherwise, it will need to be manually started by the GM later. (Timers may be manually paused and resumed as well as manually adjusted by the GM as needed.)

Show to Players —if selected, the timer is visible to the players. Otherwise, it only appears on the GM's console.

If you leave the "Targets" field blank, the timer will be considered to be applicable to everyone. If you want to indicate that only some players are affected by it, you can type their names in this field separated by spaces:[5]



This doesn't necessarily hide the timer from anyone, but players who select View » Show Timers » mine will only see timers which target their name specifically and ignore all the others, so this allows them to filter their view

[5]Since the list of names is space-separated, and usernames *should* be single words, this should be simple and convenient to type. However, if someone in your group deciced to use a login name which has spaces embedded inside it, you need to enclose their name in braces, so the value would look like "`Alice Bob {Weird user name} Charlie`". But we don't recommend using usernames with spaces in them anyway.

to only the timers they care about. Anyone who selects View › Show Timers › all will still see all the timers in any case.

To make this easier, you can also click the ME button to set the target to yourself, or the ... button to bring up a dialog box with all the currently logged-in users as checkboxes you can select from to pick the targets for your timer.

Once you're ready to submit your timer, click on the Request button.



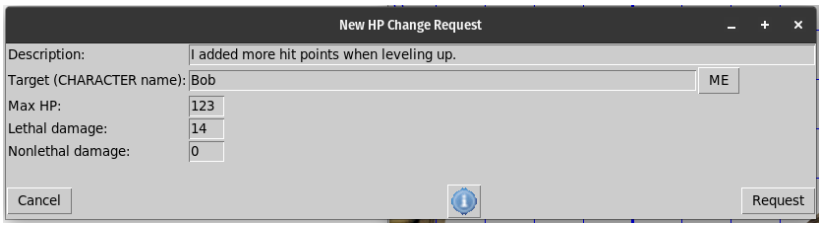
This will send your request over to the GM for review. Since they have to accept it into their time tracker (and may make modifications to it first if necessary, or might even decide for some reason not to accept it into the system), this only works if your GM is also running their GM tools at the same time you make your request, since **this is a "live" interactive operation** between the two of you.

Leave the dialog up in the mean time. If the GM sends the request back to you for you to make some kind of change to it, the dialog will update and let you make those changes and press Request again without starting over. If there's a problem, the dialog will display that as well. When the GM accepts your timer into the tracking system, the dialog will disappear automatically. (There's nothing wrong with manually clicking the Dismiss button to dismiss the dialog before that happens anyway if you choose to, but if you do that, you won't be able to see those updates and if the request isn't accepted you'd have to start over with a new one.)

Adjusting Your Hit Points

If you want to ask for a correction to the number of hit points being tracked for your character, you can submit a request to the GM to that effect using the mapper in a manner similar to what we described above to request a timer.

To start, select Play › Request Permanent Hit Point Adjustment... from the main menu. This will bring up a dialog box where you can enter your request to the GM:



You can leave a note to the GM if you wish as to why you are requesting the update. In the **Target** field, enter your character's name *exactly as it appears on the map* (including capitalization). If you logged in to the mapper with the same name as your character name, you can just click the ME button to have that name placed in this field for you.

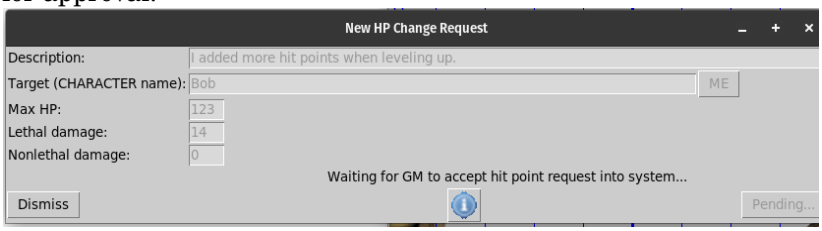
In the remaining fields, enter the requested hit point information:

Max HP: The total number of hit points your character would have if fully healed, not counting any temporary hit points.

Lethal damage: The number of hit points of damage you have suffered against your maximum hp total.

Nonlethal damage: The number of points of nonlethal damage you have currently suffered.

When ready, click the Request button to send this request over to your GM for approval.



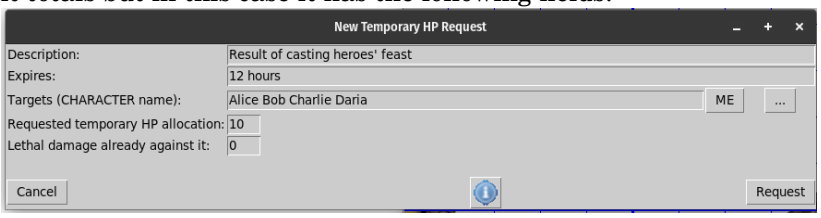
Note that, since this is a live, interactive operation between you and the GM, you both need to be logged in at the same time (the GM needs to be using the GMA program) for this tot work. The GM may adjust the numbers you requested or they may reject the request, sending it back to you for you to change it and re-submit again. If they accept it, the dialog box will close on your end.

If you don't want to wait for that, you can still click the Dismiss button to close the dialog on your side early, but this won't stop the GM from continuing to work on your request anyway.

Temporary Hit Points

Similarly, you can request that the GM add a number of temporary hit points to your character, or to a number of characters if you took an action (such as casting a spell) that provides temporary hit points to multiple creatures at once.

To start this, select Play » Request Temporary Hit Points... from the main menu. You will see a dialog box similar to the one to adjust your permanent hit point totals but in this case it has the following fields:



Description: A comment to the GM letting them know why you're requesting temporary hit points.

Expires: The duration of the effect that grants the temporary hit points. They will automatically be removed from the target character(s) after this.

Targets: List all of the names of the characters (or other creatures) to receive these temporary hit points. Note that this must be the names of the *characters* as they appear on the map (including capitalization), not the names the players logged in to the mapper tool with. These are separated from each other with spaces.

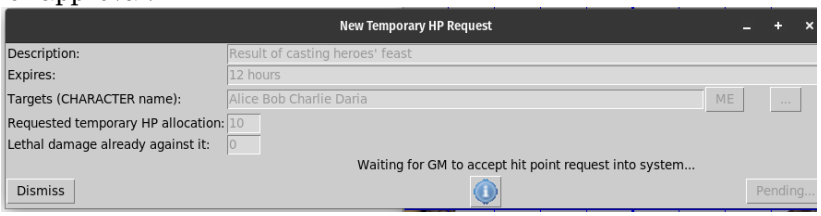
You can add your own name to the list by clicking the ME button.

You can add the names of people who are logged in to the mapper by clicking the ... button. This will give you a selection box with checkboxes for each user that you can use to select which people to include. (Again, this only works for people whose login names match the way their characters appear on the map exactly).

Requested temporary HP allocation: The number of hit points you're asking to be added to your character's "pool" of temporary points.

Lethal damage already against it: If your character has already taken damage that counts against their temporary hit points, it's best to go ahead and ask for the full original number of temporary points but also note here how many of those are already used up due to lethal damage, so both are accurately tracked by the system.

When ready, click the Request button to send this request over to your GM for approval.



Note that, since this is a live, interactive operation between you and the GM, you both need to be logged in at the same time (the GM needs to be using the GMA program) for this tot work. The GM may adjust the numbers you requested or they may reject the request, sending it back to you for you to change it and re-submit again. If they accept it, the dialog box will close on your end.

If you don't want to wait for that, you can still click the Dismiss button to close the dialog on your side early, but this won't stop the GM from continuing to work on your request anyway.

Main Menu

Even if the GM has turned off your toolbar, you can turn it back on by selecting the View » Show Toolbar menu toption.

Have Fun!

That's really all you need to know to use the mapper as a player in an active game session. What follows are some odds and ends you may find useful as well.

Hints and Tips

Rolling Percentile Dice

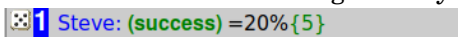
The feature described here depends on the mapper being connected to a server.

Sometimes you're in a situation where there's a percentage chance of something happening. For example, you have concealment and an opponent rolls successfully on their attack roll to hit you. Due to concealment you get to make a roll to see if the 20% chance for them to still miss you anyway saves you from harm.

The die-roller has a simple way to roll percentile dice when there's a given percentage chance of something succeeding. Just type "20%" into the "Roll:" field and press Enter.



This will roll percentile dice and report with an indication of whether the thing that had a 20% chance of succeeding actually did so

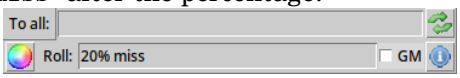


or if it failed.[6]

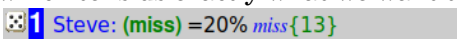
[6]The actual way the dice work here is that a d% roll (also known as "d100" since it produces a number from 1–100) is made, and a result that is less than or equal to the percentage chance of the thing succeeding means that it did. Anything over that number means failure. You can



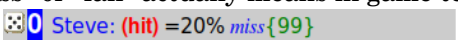
That's fine, but we can do better. In our example, where there's a 20% chance that the attacker missed due to concealment, the report from the die roller is a little counterintuitive, since here "success" means "the concealment successfully made the attaker... um... fail to hit." It would be better to tell the die roller what there is a 20% chance *of*. We do this by adding the word "`miss`" after the percentage:



Now if the miss chance was successful (20% of the time), the die roller will report "`miss`" which tells us exactly what we want to know.



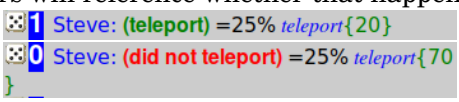
Otherwise, it will print "`hit`" so there's never any misunderstanding about what "success" or "fail" actually means in game terms.



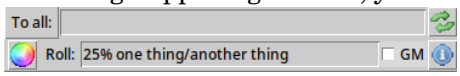
These can be anything you like, not just about hitting or missing a target. If for some reason there is a 25% chance that a creature will teleport, just roll for "`25% teleport`":



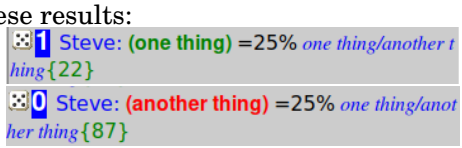
And the answers will reference whether that happened or not:



If you want a specific message to be used for the success and failure cases, just put the success message first, then a slash, and the failure message. For example, if there's a 25% chance of one thing happening, or a 75% chance of another thing happening instead, you can do this:



With one of these results:



For the full story of all the things you can do with the die-rolling feature of the mapper, see Chapter 10 starting on p. 187.

Formatting Chat Messages

Starting with Mapper version 4.29, you can use GMA markup formatting codes in your chat messages. This is explained more in Chapter 10 (p. 191).

see the actual die-roll results in the output shown in these examples, as well as the number 0 reported as the overall die-roll result in case of failure, and 1 as the result in case of success.

Epilogue

Manpages

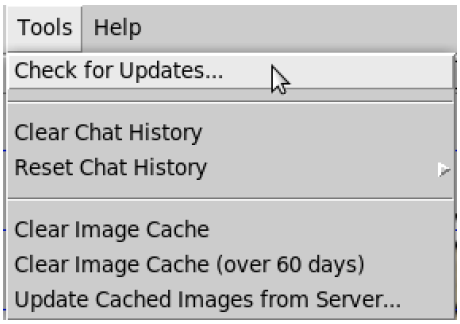
Detailed documentation about the mapper (but not necessarily as friendly as this tutorial) may be found here, including many more things you can configure to change its appearance and behavior:

- Operational guide and description of the network protocol (page 470).
- Description of the map file format (page 426).
- Syntax used for making die rolls (page 621).

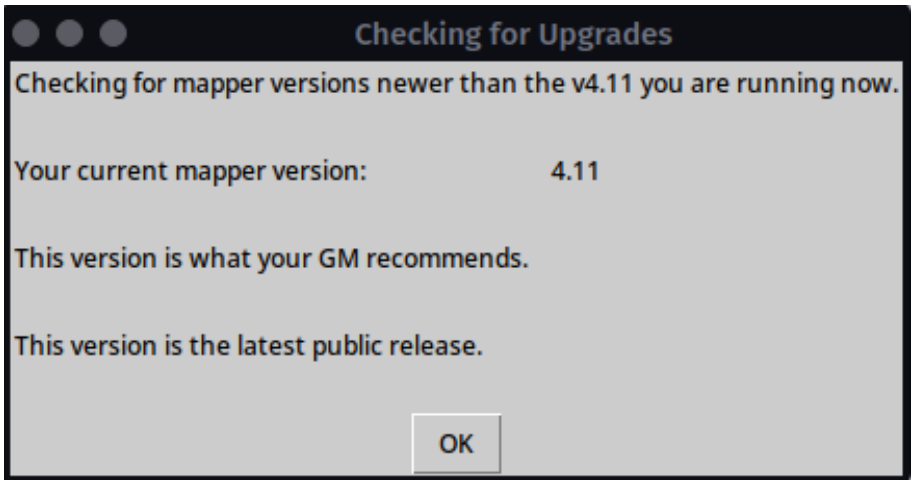
Upgrading Your Mapper

It is important to keep your mapper up to date so you get the latest features and bug fixes. If your GM has configured the game server to recommend a specific version of the mapper for everyone in your group to use, you will automatically be informed if your mapper is a different version from the one your GM recommends. When this happens, you will also be given the opportunity to upgrade your mapper.

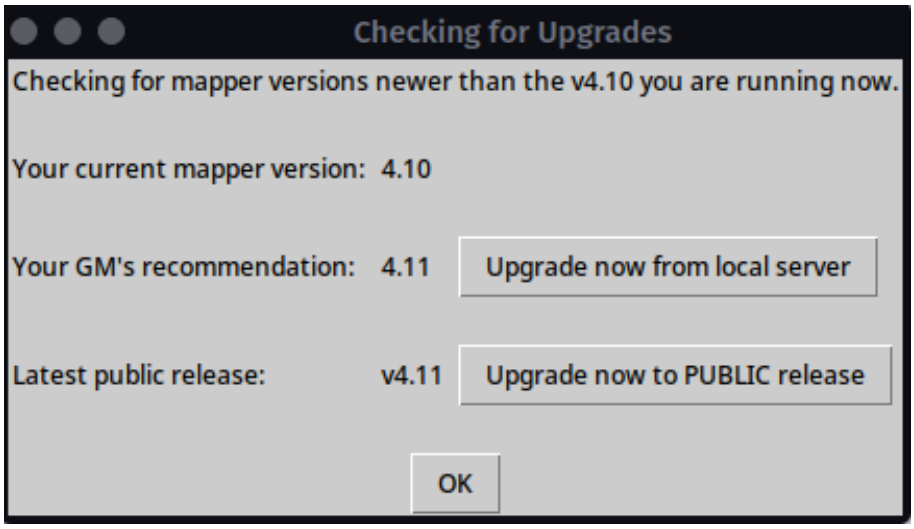
You may manually check for updates by selecting Tools » Check for Updates... from the main menu.



If your Mapper is already up to date, you will see confirmation of this fact:



Otherwise, it will tell you if you what updates are available to bring you up to date with your GM's recommendation and/or the latest public release.



Click on the Upgrade now from local server button to download and install the version of the mapper your GM recommends. This may come from your GM's game server or from the public repository, depending on how your GM has configured the server and your client.[7]

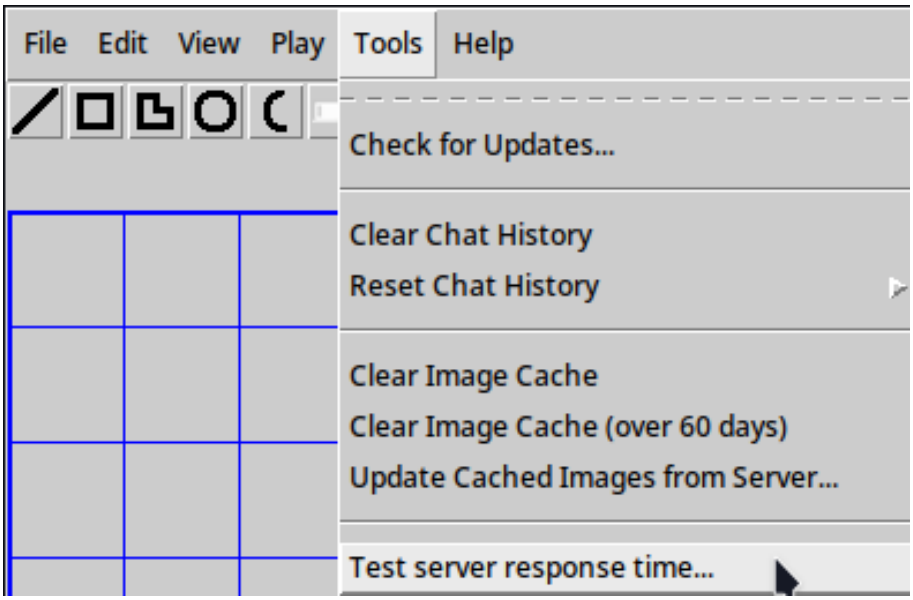
Click on the Upgrade now to PUBLIC release button to retrieve the latest public release of the mapper program from the public repository. This might

[7]Note that if the GM's configuration has you set up to retrieve updates from the public server (github), only the most recent several versions are available there. If their recommended version lags too far behind, the upgrade will fail because your mapper won't be able to find the outdated version anymore. In that case, the GM will need to offer a copy of the mapper on their own server, or—better yet—recommend a newer version for you to use.

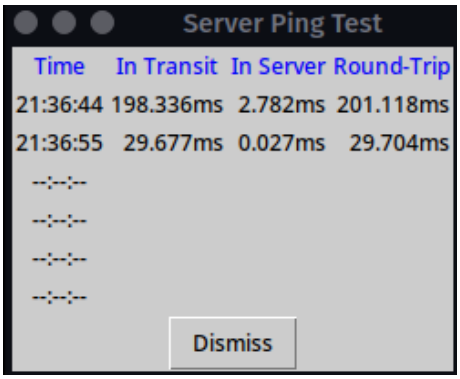
be newer than the version recommended by your GM if they aren't keeping that recommendation up to date.

Testing Server Response Time

If the communication with the game server seems to be sluggish, you can open a tool in your mapper to get some insight into the issue. Select Tools 〉 Test server response time... from the main menu.



A small window will open which displays response times. Every 10 seconds, the mapper will send a small "ping" message to the server and measure the time it takes for it to echo back to the mapper again. This is displayed as four numbers for each ping. The first is the local time for when the ping was initiated. The next is the length of time the message took to travel through the network between the mapper and the server. The next is the length of time it was being processed inside the server itself. The last is the total round-trip time (which is the sum of the previous two).



If the round-trip and in-transit times are high, that usually means you have a slow or overloaded network connection somewhere between you and the server (not necessarily your local network itself). If the in-server time is high, that may indicate excessive load or other problem with the server itself.

The tool will continue to ping the server every 10 seconds until you click the Dismiss button to close the window.

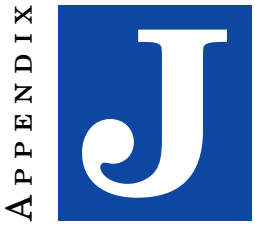
Diagnosing Problems

The mapper will open a diagnostics window when it needs to report error messages (as well as warnings and informational messages). If you need to see more debugging output, you can adjust the values in the preferences editor's Diagnostics tab.

If you wish to save some of these diagnostic messages to report a bug or seek help with solving a problem, you can select a range of text in the diagnostic window and copy/paste into another document or program. You may also save the entire contents of the window to a text file via the Tools › Save diagnostic messages as... menu.

Missing Server Images

If your GM has loaded images on the server for your mapper client to use, but for some reason the images couldn't be retrieved (such as, say, due to the GM configuring the server to think the image is at a particular URL on the server but that file is missing), then the mapper will stop asking again for that image to avoid spamming the server with requests we already know won't succeed. If the GM then fixes the problem, you can select Tools › Retry failed image retrievals from the menus to tell the mapper to allow another shot at all the images it failed to retrieve since it was started.



Server Quick Reference

> The X server has to be the
> biggest program I've ever seen
> that doesn't do anything for you.
>
> —Ken Thompson

The following guide can be given to whomever will set up and maintain your GMA server. It is a concise, condensed version of the material written in more detail elsewhere in this manual, collected here in a single few pages for easy reference.

This is not, therefore, the whole story but just the highlights. Please also refer to:

- The server program documentation in `server`(6) on page 597,
- The server check program documentation in `servercheck`(6) starting on page 545,
- Chapter 9 starting on page 171,
- Chapter 10 starting on page 187,
- Chapter 11 starting on page 209, and
- The protocol definition in `gma-mapper-protocol`(7) starting on page 627 if you need to have a look at the internals.

Installing the Server

You will need a server your players can reach over the network. We recommend a Linux or BSD system, but in theory it should be possible to get most systems to host your GMA server. We assume anyone setting up and maintaining such a system has the necessary system administration and security skills to do this correctly. This is not a guide for someone new to that field.

Prerequisites

You need the most recent version of the Go compiler on your system if installing and building from sources.

Cloning from Github

Clone the repository to your server system into the location of your choice. We'll use `/opt/gma` for this reference guide.

```
$ cd /opt/gma
$ git clone git@github.com:MadScienceZone/go-gma.git
Cloning into 'go-gma'...
remote: Enumerating objects: 2532, done.
remote: Counting objects: 100% (806/806), done.
remote: Compressing objects: 100% (235/235), done.
remote: Total 2532 (delta 541), reused 786 (delta 529), pack-
reused 1726
Receiving objects: 100% (2532/2532), 3.37 MiB | 6.21 MiB/s, d
one.
Resolving deltas: 100% (1704/1704), done.
```

Compiling

Run the `make` (or `gmake` if that's available or your `make` command is too old to understand the `Makefile`) from the repository root to build all the programs, including the server.

```
$ cd /opt/gma/go-gma
$ make
building map-console
building map-update
building preset-update
building server
building upload-presets
(cd man && make)
make[1]: Entering directory '/tmp/go-gma/man'
(cd man6 && make)
make[2]: Entering directory '/tmp/go-gma/man/man6'
make[2]: Nothing to be done for 'all'.
make[2]: Leaving directory '/tmp/go-gma/man/man6'
make[1]: Leaving directory '/tmp/go-gma/man'
```

Running the Server

The server may be run directly out of the repository, or you may move the `go-gma/cmd/server/server` binary to another location and run it from there. Assuming a typical set of configuration options and files were set up as described in the following sections, the server's invocation would be

```
/opt/gma/go-gma/cmd/server/server -endpoint :2323 \
   -init-file mygame.init -log-file mygame.log \
   -password-file mygame.sec -sqlite mygame.db \
   </dev/null >>mygame.out 2>&1 &
```

Figure J.1 is a sample script one might find convenient to start a GMA server on a system hosting several such servers. A "start all servers" script such as that shown in Figure J.2 could be used to start up all the servers you're hosting.

With that setup, a script such as the one shown in Figure J.3 can be run periodically to force the servers to restart, which in turn will cause the server start script to compress and archive their old logfiles and then start a new copy of the server, writing to new log files.

Configuration

Set up an initialization file (e.g., `mygame.init`) with the contents shown in Figure J.4. Set up the authentication file (e.g., `mygame.sec`) with the contents shown in Figure J.5, except for filling in your own desired passwords.

```
#!/bin/sh
# Start a GMA server, and re-launch when it exits.
# Usage: start-server SERVERNAME PORT
BASE="$1"
PORT="$2"
if [ "$BASE" = "" -o "$PORT" = "" ]; then
    echo "Usage: $0 base port"
    exit 1
fi
DEBUGOPTS='none'
while true; do
    if [ -f "$BASE.stop" -o -f "ALL.stop" ]; then
        echo NOT restarting $BASE server
        exit
    fi
    TODAY=`date +%Y%m%d`
    echo "-----[ Launching on $TODAY ]-----"
    for suffix in out log; do
        for oldfile in $BASE.*.$suffix; do
            if [ "$oldfile" != "$BASE.$TODAY.$suffix" ]; then
                gzip -v $oldfile
                mv $oldfile.gz old-logs
            fi
        done
    done
    /opt/gma/go-gma/cmd/server/server -endpoint :$PORT \
        -init-file $BASE.init -password-file $BASE.sec \
        -sqlite state/$BASE.db -debug $DEBUGOPTS \
        -log-file "${BASE}.${TODAY}.log" </dev/null \
        >>$BASE.$TODAY.out 2>&1 &
    SERVER_PID="$!"
    echo "$SERVER_PID" > $BASE.pid
    wait "$SERVER_PID"
    rm $BASE.pid
done
```

Figure J.1: Example Server Startup Script

```
#!/bin/sh
start_server MYGAME1 2323 </dev/null >>MYGAME1.startlog 2>&1 &
start_server MYGAME2 2324 </dev/null >>MYGAME2.startlog 2>&1 &
```

Figure J.2: Example Script to Start All Servers

```
!/bin/sh
# Rotate server log files for map servers.
# This requires that we identify and stop the servers so we
# start writing to new files All we need to do is send the
# servers an INT signal. This will shut them down, and their
# start_server scripts will compress old logs and start them
# up again.
cd /opt/gma
# list the base names of your game servers where MYGAME1,
# MYGAME2, etc are shown here.
for base in MYGAME1 MYGAME2; do
   if [ -f $base.pid ]; then
      echo "Stopping $base server..."
      kill -INT `cat $base.pid`
   else
      echo "$base server does not appear to be running."
   fi
done
```

Figure J.3: Example Log Rotation Script

```
// Unauthorized access prohibited
// My GMA Server
AUTH
WORLD {
   "Calendar": "golarion",
   "ClientSettings": {
      "ImageBaseURL": "https://www.example.com/gma"
   }
}
SYNC
```

Figure J.4: Example Server Initialization File

```
PLAYERS-SHARED-PASSWORD-HERE
GM-PRIVILEGED-PASSWORD-HERE
USER1:PASS1
USER2:PASS2
```

Figure J.5: Example Server Authentication File

If you have users who want to have their own personal passwords, add them in place of the USER1 and USER2 sample entries, and so on, one per user. Otherwise, you only need the first two lines.

Listing Party Members

For each party member or significant NPC who should be listed in the mapper's main context menu, add an AC line to mygame.init, with appropriate unique ID values and other attributes. For example:

```
AC {"ID":"PC82", "Name":"Alice", "CreatureType":2, "Size":"M",
    "Color":"blue"}
AC {"ID":"PC85", "Name":"Bob", "CreatureType":2, "Size":"M",
    "Color":"blue"}
```

Setting Custom World Attributes

If you use a different calendaring system than Golarion, adjust the WORLD line accordingly. You may also add custom creature condition markers as documented in the server(6) documentation starting on page 597.

Mapper Client Versions

When your players connect their mapper clients to your server, it can tell them what version of mapper you want them to use, to keep everyone on versions compatible with each other and with your server. To do this, add a configuration entry in your mygame.init file like shown in Figure J.6.

These settings, and the "Mapper Upgrade URL" mapper setting referenced below, will cause mapper upgrades to be pulled directly from the Github server. Only the most recent several releases are available there. If you want to stay at an older version or want total control of the distribution of your players' mapper clients, you can set up a server of your own to hold the signed mapper release files and have your server and client configurations changed to reference the server and filenames of your choice. See the full server(6) documentation (p. 597) for details.

```
UPDATES {
   "Packages": [{
      "Name": "mapper",
      "Instances": [{
         "Version": "4.6",
         "Token":   "mapper-4.6"
      }]
   }]
}
```

Figure J.6: Server Configuration for Automatic Updates

Ongoing Maintenance

When a new version of the server is released, you should upgrade to it by running `git pull` in your cloned `go-gma` repository, then running `make` as described above. Restart the server binary.

When a new version of the mapper client is released, you should update `mygame.init` to show the new version. Then either restart the server or send its process a `USR1` signal to cause it to reload the `mygame.ini` file. Now when player clients connect, they will be directed to upgrade.[1]

Servercheck Script

You should arrange to run `servercheck`(6) periodically on your server to check on the status of your server, alerting you in case it is down or deadlocked. See page 545.

Setting Up the Web Server

If you want to serve graphic files to your players' mapper clients (and you probably do), you'll need to set up a web server to host them. For example, on your `www.example.com` server, set up a document area that can be reached by a URL starting with `https://www.example.com/gma`. This is the base URL given to GM utilities such as `gma rendersizes` (see p. 532).

Make sure your mapper client settings include the correct paths to reach this area via `ssh` and `scp` so your client can upload content to this document directory tree on the web server during the game.

[1] Note that you can't alter the files. They must be the `*.tar.gz` and `*.tar.gz.sig` files for the given version of the mapper program as distributed in the `signed-releases` directory of the repository. The mapper won't automatically upgrade to a file that doesn't have a correct cryptographic signature.

Setting up the Web-based Die Roller

To provide the simplest possible way to let your players make die rolls with the GMA server (without needing to install the mapper client or any other client program on their system), simply copy the files from `extras/webroller` to a directory on your web server. Also copy the file `man/man7/gma-dice-syntax.7.pdf` into that same diretory.

For example, you install them in a directory which can be seen from the internet by the URL `https://www.example.com/gma/webroller`, then if your players go to that URL they will be presented with a simple page which provides this access. It allows most (but not all) features of the one built-in to the mapper client.

This requires an adapter to provide websocket-compatible access to the GMA server in order to work. This can be done by obtaining the `websockify` program, and then running it with a TCP port that's not currently in use on the server. For our example, since the GMA server is running on port 2323, we'll run the websockets interface to the GMA server on port 2324:

```
$ websockify --daemon 2324 127.0.0.1:2323
```

If your site uses SSL, you may need to supply the server's encryption key and certificate chain so the websockify proxy can be trusted by the browser. This may also require it to be run with elevated privileges so it can read the encryption key file:

```
# websockify --daemon --cert=/etc/ssl/example.cert \
     --key=/etc/ssl/example.key 2324 127.0.0.1:2323
```

What to Tell Your Players

You'll need to inform your players about what connection settings to use for your sever. If it is set up as described above, on a server called `example.com`, they should make the following settings in their mapper preferences:

Hostname: `example.com`

TCP Port: 2323 (and 2324 for the web die roller)

User/Character Name: (whatever they choose)

Password: (their personal password or the shared players' password)

Server Image Base URL: `https://www.example.com/gma`

Mapper Upgrade URL: `https://raw.githubusercontent.com/MadScience-Zone/gma-mapper/main/signed-releases`

Using the Server

The `map-console`(6) program (see p. 581) is a convenient way for you to manually interact with your sever as needed, although in the normal course of preparing and running a game you should be able to accomplish what you need using just the `gma` and `mapper` programs.

Use `rendersizes`(6) (p. 532) with the `-w` option ahead of time to prepare images for creature tokens and map image tiles. This produces a `.map` file and a `gma` directory tree full of files as they should exsit on the web server. If you use `scp -r` to copy the contents of the `gma` directory to your `gma` document directory oin the web server, they'll be where the server expects to find them later. From your mapper client, load the `.map` file that was produced by `rendersizes`. This will inform the server to associate the web server files with the image names as referred to in the maps. The server will store that information it its local database and will pass that information on to mapper clients as they go looking for images.



Cryptographic Signatures

> I understood the importance in principle of public key cryptography but it's all moved much faster than I expected. I did not expect it to be a mainstay of advanced communications technology.
>
> —Whitfield Diffie

The official releases of the mapper client are available in ZIP and tar formats on the github repository `MadScienceZone/gma-mapper`, in a subdirectory called `signed-releases`. The mapper is able to automatically upgrade itself by downloading those files. To guard against installing a corrupt file, the automatic upgrade code in the mapper first checks the digital signature of that file.

If you have a need to manually validate the signature on copies of these release archives, we provide the public key necessary to perform that validation here.

```
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAljm0271r46IM1W3KsN
koKoq++ysns6Gnp5nZLGLqzuhV8ifF3uml12nfO3UPS6vEzD5j//j0iEmchQEe
HzuyjZUDyox4fM4lJ5oQBw2/UU3RBtoR3t/HlyULKvfJf46TRKehKcuTJYqC4F
Xn7CzS77OXka2Z/0flwGN/iGFZoG3hH77bLhu5i61hSx4NkTPd0gWfVcNOUI8p
bQHngV0xdwsnoRkiKTnwyPolQn43eUKP/ErM3QmkUdztsRdGoMdC6ukLDXCXOB
EVnu4VJCQ+i/2iUa06anlVPcOy/HrUra/0+KUdXHn/zu9IaD9z6Ll5XRoujSGK
VcEiPeZFiPd1tgvsz1eZiN6wTKqxqU43LG7eNGO2ggaoLtF2ABOpC/91wtQ1/8
px5qvenQIBJAqRPJ7+Af7eyUcQ3neske8I/PfphutOXP0Zj5ypNXDiSxw2qJ3u
8erXmRZ4Ph0c9c9m6KL3+9KYYoQ4nQ/CiBwjf6GKMz3NEXQQF9IMtbAcKEp2lR
tVNZ/AyFficx6W3GRkZ0bSBVrgel+iZQjjKJ2osOoyhAQQp/P5RQQPupgmxkvn
dc/9gWYE6JX1YJUXCOqLU64greY1BqGzvJuY4UzIy+1/+4ZPjhYzxBwkbrsW6d
ZNvsq8zQyK+KxLQiueGzKI7dPxqfF/kEZTWVPJMOX07OUCAwEAAQ==
-----END PUBLIC KEY-----
```

Colophon

This manual was composed and typeset by the author using LaTeX with Memoir layout macros, augmented by wrapfig, lettrine, wallpaper, TikZ, and a host of miscellaneous behind-the-scenes working packages and scripts.

It was set 10/12 pt using the TeX Gyre Schola font family created by GUST, the Polish TeX User Group. This typeface is based on URW Century Schoolbook L, originally designed by Morris Fuller Benton in 1919, for the American Type Founders.

Published electronically in PDF format for ease of viewing on any platform.

Art Credits

Front Cover

"Dungeons and Dragons game" by Rocco Pier "Moroboshi" Luigi (Wikimedia Commons). Used as allowed by owner's Creative Commons attribution share-alike license.

Mapper Tutorials (Chapter 5 and Appendix I)

- "An Inn With a Secret" by Jonathan Roberts (http://www.fantastic-maps.com/free-stuff/). Used as allowed by owner's Creative Commons attribution non-commercial share-alike license.
- "The Caves of Chaos" by Mike Weem. (http://www.theweem.com/2012/-02/caves-of-chaos-reimagined-by-weem/.) Free for noncommercial use.
- "Vulcan" by Deevad (https://deevad.deviantart.com/art/Vulcan-30291-6233). Used as allowed by owner's Creative Commons attribution share-alike license.
- "Wizard of Leandriss Church" by Igor Esaulov (https://www.deviantart-.com/igor-esaulov/arg/Wizard-of-Leandriss-Church-295619526). Used as allowed by owner's Creative Commons attribution non-commercial share-alike license.

Dénouement

If you really want to know why we use "//" rather than "/" for division in die roll specs (see p. 192), (1) it is patterned after Python's "//" operator to denote integer division, (2) there are already other contexts in the die-roll expressions which use "/" so it's easier to just split apart the die-roll expression on "//" initially before parsing the individual pieces which may have "/"s in them, and (3) it avoids confusion with the usage of "/" as an alternative marker for things like multiple attacks or the `{...}` notation.

Unicode-aware programs can avoid this issue entirely by using the "÷" character instead of "//" for division in die roll specs.

Index

.gmo file, *see* encounter sheets
.profile, 12
//, rationale for, 794
/opt/gma, 6, 782
$GMA, 6
$GMA/bin, 6, 12, 13
$GMA/lib, 6
$GMA_IGNORE_GO_GMA, 12
$GMA_PERL, 12
$GMA_PYTHON, 12, 14
$GMA_WISH, 12
$PATH, 12, 13
SSH certificate, 10
~/.gma, 14

actions
    tracking of, 159
AD&D, 405
Amiga, 8
Analects, The, 573
API, 4–6, 227, 228, 573
apt, 9
art credits, 793
authentication, 174, 181, 182, 783

Blood, Jen, 97
Bon Jovi, Jon, 159
breath weapon
    recharge tracking, 170
BSD, viii, 717, 718, 782

calendar, 21, 46
caster level, 31
Cat Who Walks Through Walls, The, 613
Caves of Chaos, 83
character record sheets, 97–121
    file format, 99
chat channel, 187–190
CL, 31
CLI, 223, 449, 573, 726, 728, 729, 735–741
clock
    absolute time, 22
    adjusting, 22
    elapsed time, 22
    free-running mode, 22
    setting, 21
cobblestone, 82
combat, 22, 32–44, 155, 746
    creatures re-entering, 155
command spell, 223
condition timers, 44, 177
conditions, 3
Confucius, 573
counters, 2, 28–32
creature tokens, 73, 179
    conditions, 753
    killed version, 77
    transparent, 81
creatures, 3
curl, 10

d20, 4
D&D, 1, 4, 83
DC, 194
Deevad, 793
die roll expression, 192
die roll presets, 196
die roller, 192–208
    automation, 207
    delegates, 205

in-browser, 206
Diffie, Whitfield, 791
disease
tracking saving throws, 162
Dongo, Studio, 1
Donuttus, 46
drains, 29–32
Dumbledore, Albus, 375

encounter sheets
attack rolls, 146
consistent randomness, 142
critical damage, 147
cultures, 138
damage rolls, 146
duplicate names, 129
files, 123
generic names, 138
hit points, 142
layout, 127
monster groups, 131
monster names, 131, 134
monster templates, 144
shapechanging, 147
spawn points, 152
verbosity, 158
environment variables, 12
Esaulov, Igor, 75, 793
EULA, 710
events, *see* timers

Fields, Bryan, 73
file formats, 375
flat-footed, 37, 38, 40, 177
FreeBSD, 5, 7, 8, 220, 717, 720–722, 726

Game Master, *see* GM
Ghostscript, 10
GIF, 74, 79, 213, 728
GIMP, 74
git, 8, 10
Github, 218
GM, 1–3, 5, 9, 14, 16, 17, 22–24, 26, 31, 32, 38, 39, 49, 50, 84, 88, 97–99, 113, 116, 118, 123, 125, 127, 139–142, 148–150, 153–155, 159, 171–175, 180–183, 189, 190, 195, 201, 203, 207–209, 211, 215–217, 628, 708, 714, 718, 724, 725, 728, 737–739, 745, 748, 756, 763, 764, 769–775, 777–780, 787
GMA, rationale for, 1
Go, 8–10, 573
Go-GMA, 10–12, 171, 172, 183
Golarion, 21, 46, 47
GUI, 124, 720, 728
GUST, 793
Gygax, Gary, 13, 171, 187, 209, 449, 725

health bars, 177, 748
blurring, 181, 748
health trackers, 3, 36–38
Heinlein, Robert, 613
held actions, 40
Hero Lab, 18, 98, 118
Homebrew, 9, 720
HP, 35, 37–40, 42, 123, 143
HTML, 118

ID, 94, 98, 99, 113, 124, 216, 739
image tags, 126, 131, 136
ImageMagick, 10, 74
INI, 99, 105
initiative count, 22
initiative list, 32, 38, 176
saving, 46

JPEG, 74, 213
JSON, 629, 630

Keep on the Borderlands, the, 83
Kreeft, Peter, 717

LGM, 139
Linux, 7, 8, 717
Lone Wolf Development, 98
Luigi, Rocco, 793

macOS, viii, 5, 7, 9, 74, 84, 717, 718, 720, 726
man, 223, 449, 573
manpages, 223, 449, 573
map, 3
map-lock, 70
map-translate, 71
mapper, 9, 11, 12, 49–71, 73–94, 725, 791
  arc tool, 54
  area of effect, 751, 764
  cache, 217
  chat channels, 762
  circle tool, 54
  clear buttons, 60
  color selection, 58
  combat, 62, 746, 763
  conditions, 753
  configuration, 726–740
  console client, 183, 789
  delete tool, 57
  die rolling, 762
  distances, 758
  exit button, 60
  file operations, 70
  grid lines, 60
  grid snap, 59
  gridlines, 758
  image tiles, 82, 209
    server ID, 213
  incremental reveal, 70, 745
  line thickness, 60
  line tool, 52
  load buttons, 60
  location marker, 758
  map files, 80
  move tool, 57
  movement modes, 752
  play mode, 57
  polygon tool, 53
  pre-made maps, 83
  prescaling images, 88
  reach weapons, 750
  rectangle tool, 53
  save button, 60
  server, 171–185, 781–789
  server connection, 744, 788
  server protocol, 613
  server security, 211, 783
  signing key, 791
  stamp tool, 57, 82
  store-and-forward, 216
  text tool, 55
  threat zones, 177, 747
  tool bar, 742
  unload button, 60
  upgrades, 217, 787
  zoom buttons, 60
mapper, 10
maps, 3, 49
markup symbols, 128
Mercer, Matthew, 7
miniatures, 9, 49
mkmap, 9, 49
modules, 83
Monty Python, 159

namegen, 134
NPC, 27, 49, 60, 98, 123, 124, 145, 151, 786

OS, 9

package managers, 8, 10
passwords, 174
Pathfinder, 4, 46, 47, 97, 725
PC, 2, 17, 29, 44, 60, 98, 123, 124, 130, 145, 151, 153, 175, 183, 209, 747
PDF, 10, 94, 96, 101, 115, 793
Perl, 8, 9, 49
Photoshop, 74
pip, 9
PNG, 74, 79, 213, 728
poison
  tracking saving throws, 162
Porter, Louis, Jr., 227
POSIX, 5
PostScript, 10, 101, 613
Pratchett, Terry, 49
Preview, 74
printmon, 126, 128

progress clocks, 159
progress tracking, 159
protocol, 613
`ps2pdf`, 10, 125
PSP, 2
public key, 791
Python, 8, 9, 14, 720

readied actions, 40
`rendersizes`, 9, 10, 77, 79, 82–84, 88, 94, 789
resource pool tracking, 167
Roberts, Jonathan, 793

`scp`, 10
shell scripts, 8
SOCKS5, 738
sqlite, 173
SSH, 10–12, 212
SSL, 736, 788
stabilized, 37
SunOS, 7

tar, 791
Tcl, 8, 9, 720
`tcllib`, 9
TCP, 172, 173, 737, 788
Terrors of Obsidian Apocalypse, 227
Thompson, Ken, 781
Tian Xia, 47
time tracking, 2, 21
timers, 2, 23–27
  public, 26
Tk, 9
Tk libraries, 720
`tkinter`, 9
`tkinter-tooltip`, 9
`tklib`, 9
toolbars, 175

Unicode, 203, 794
Unix, 7, 8
URL, 182, 211, 214, 218, 219, 221, 739, 780, 786–788
UUID, 99, 100
`uuidgen`, 99

VMS, 8

Waterhouse, Agnes, 74
Weem, Mike, 83, 793
Windows, 8, 707
witchcraft, 74
wounds, 39

XML, 118

`yum`, 9

ZIP, 217, 708, 715, 718, 791