NAME

gmaproto – GMA client/server communication functions

SYNOPSIS

(If package installed globally)

```
package require gmaproto
```

(Otherwise)

```
source gmaproto.tcl
```

`::gmaproto::add_image` *name sizes* ?*frames*? ?*speed*? ?*loops*?

`::gmaproto::add_obj_attributes` *id attr values*

`::gmaproto::adjust_view` *x y grid_label*

`::gmaproto::allow` *features*

`::gmaproto::auth_response` *challenge* ?*passes*? → *response*

`::gmaproto::background_redial` *tries*

`::gmaproto::character_name` *names*

`::gmaproto::chat_message` *message sender recipients to_all to_gm* ?*markup*? ?*pinned*?

`::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* ?*globals_only*?

`::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* ?*roll_id*? ?*targets*? ?*roll_type*?

`::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* ?*frames*? ?*speed*? ?*loops*?
: 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). If given, *frames*, *speed*, and *loops* indicate how to animate the frames of an animated image. 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 grid_label*
: 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. If *grid_label* is nonempty, the view is adjusted so the grid with that label is in view. 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* ?*passes*?
: *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::character_name` *names*
: Declare that the player is controling the character(s) whose names (as shown on the VTT map) are as listed in the list *names* instead of the name under which the player logged in to the `mapper` client.

`::gmaproto::chat_message` *message sender recipients to_all to_gm* ?*markup*? ?*pinned*?
: 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*. If *markup* is true, GMA markup syntax is honored in the chat message. If *pinned* is true, this message will be pinned for long-term display. 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* ?*globals_only*?
Asks the server to remove all stored die-roll presets whose names match the regular expression *regex*. If *globals_only* is true, this only applies to system-defined presets. 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* ?*roll_id*? ?*targets*? ?*roll_type*?
: 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*. If given, *roll_id* assigns an ID to the roll which is sent back with the results. If given, *roll_type* gives the type of die roll being made. The creature names being targeted is given, if applicable, by the *targets* list value. 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.

COPYRIGHT

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.