Gnomegg: A blazing-fast third-party destiny.gg chat server

Dowland Aiello

April 17, 2020

Contents

1 Data Schematic **2**
1.1 Events 2

1 Data Schematic

Gnomegg is entirely backwards-compatible with the destiny.gg chat, and uses capnproto. Cap'n proto is generally regarded as the successor to Google's protobuf language, and is both platform and language-agnostic. Capnproto poses several benefits over protobuf. Namely:

- Cap'n Proto serves as both an interchange and in-memory storage format—no serialization necessary!
- *Promise pipelining*: subsequent messages relying on each other can be "squashed" into one message
- Cap'n Proto data can be transferred via a standard bytestream, and can be encrypted in an equally straightforward manner

Outlined below are each of the data types represented in Cap'n proto, for use in gnomegg.

1.1 Events

In gnomegg, an event represents some action taken by a user that must be handled by the server, or, in some cases, a response from the server sent to a chatter. Gnomegg uses all of the events commonplace in the destiny.gg chat server software, but adds an **Error** event type. Gnomegg's event types, alongside their respective fields are defined as such:

- issueCommand: a command was issued by a client
  - Issuer: the username of the chatter issuing the command
  - Type: a union with the following possible values representing the type of command being issued:
    - Message: an object defined as such, representing a simple text message sent by a chattter, rendered by the client:
      - Contents: the contents of the message, represented by a UTF-8 encoded string
    - PrivMessage: an object defined as such, representing a message sent only to one chatter, rendered by the client:
      - Concerns: the username of the chatter that the message should be sent to
      - Message: a mesage literal as defined above in the Message schematic
    - Mute: an object defined as such, used to mute a user for a specified amount of time:
      - Concerns: the username of the chatter who will be muted
      - Duration: the number of nanoseconds for which the targeted user should be muted, expressed as a 64-bit unsigned integer
    - Unmute: an object containing a string representing the username of the user who will be banned
    - Ban: an object defined as such, used to ban a user for a particular stretch of time:
      - Concerns: the username of the chatter banned as a result of this command's execution
      - Reason: a string representing a moderator's reasoning behind the ban command's issuance
      - Duration: the number of nanoseconds for which the targeted user will be banned from chatting
    - Unban: an object defined as such, removing a user from the ban list
      - Concerns: the username of the chatter who will be unbanned
    - Subonly: an object defined as such, making the chat sub-only mode:
      - On: whether or not the chat should be in sub- only mode
    - Ping: an object defined as such, initiating a pint- pong response loop:
      - InitiationTimestamp: a timestamp expressed as a slice of bytes, representing the time at which this command was issued
- pong: the server is responding to a client request to ping with a pong

  - Timestamp: a slice of bytes representing the time at which the request to ping was received

- broadcast: the server is broadcasting a message sent by a client to the users in the chat session
  - Sender: the username of the chatter sending the message
  - Message: the message sent, represented as a UTF-8 string

- error: the server is responding to an individual request with an error that occurred at runtime
  - Concerns (all | user (username)): the scope of the error— should it be sent to all users in chat, or just one user?
  - Error: a string representing the error that will be communicated to connected clients