Formatting

Message formatting is an important aspect of communication on an instant messaging platform. Moreover, including various options for communication (i.e., allowing images, invite buttons, or other multi-media representations) is an extremely integral aspect of any of these services. Message formatting may be done by passing a format body (JSON) alongside a message sent. The Delegate server instance shall make minimal regulations or establishments on these; it is on the sole behalf of the clients to articulate and interpret these message formattings. This section will define a common protocol for doing such.

The format body shall be an object with the range of the formatters' influences being specified within the string, which will be delimited by a "-" (hyphen) (e.g., "0-9" refers to the 0th character of the message to the 9th character). Each range then shall be subject to an object of formatters which shall add format to the message, which may include colorization, font size, boldness, or of something else. A range which encompasses the entire text—or, rather, the lack thereof, if there is no underlying text message—shall be a mere *null*.

The protocol recommends that any server implementation limits message formatting to 256 bytes in size. Certain formatters may be cataloged in messaging databases so that one may search via a specific formatting type (e.g., if a message has an image).

This method of formatting text is extremely useful because it is extremely simple for clients to implement, as all that is needed is simple processing of JSON and understanding the ranges of text to modify. It is also respectful that certain clients may be minimal such that in-text formatting (e.g., that of Discord) may not be processed and therefore clutter messaging. For example: ```c hello```. Above all, it standardizes a common language for clients.

Message formatters will have properties on themselves (e.g., a font-size requires a size, a text-color requires a color, an image requires a URL, etc). A formatter shall resolve to its properties like so:

```
{
        "color": "red"
}
```

We shall define the various message formatters, their properties, and the acceptable values of those properties below:

color — Colorize text subjectively (the client shall decide what is, for instance, red).

- **0 - red**
- **1 - blue**
- **2 - green**
- **3 - cyan**
- **4 - yellow**

- **5 - magenta**
- **6 - pink**
- **7 - purple**
- **8 - orange**
- **9 - brown**
- **10 - opaque**

- **It is to be noted that colors should be limited and that the client is tasked with correlating the names of the colors with specific hex codes.**

italics — Make text italics.
bold — Make text bold.
strikethrough — Make text strikethrough.
underline — Underline the text.

vital — Subjectively declare text to be of uttermost vitality.

- **Client's discretion.**

emphasize — Subjectively emphasize text.

- **Client gets to decide this!**

drownout — Subjectively decide what to drown out.

- **Again, the client gets to decide this!**

super — Supertext. Provide what should be above.
sub — Subtext. Provide what should be below.

image — Tell the client to display an image. Provide a URL.

- **As previously stated, message formatting is not regulated by the server, so the client must receive safe URLs from *safelinks* or from its own table and obtain the image once it is decidedly safe. If the client refuses to implement these measures, an IP address could be leaked.**

video — Tell the client to display a video. Provide a URL.

- **Same note as above applies and hereafter all formattings including a GET request to a file.**

file — Represent a file to be downloaded. Provide a URL.

- **Again, same note to be heeded.**

channel — Represent a channel. Provide a channel name.
user — Represent a user. Provide a username.
quote — Represent a message from somewhere. Provide an ID (Delegate Snowflake ID) of a message.

link — A hyperlink. Provide the URL.

- **The client should receive the website metadata on its own discretion. Some clients may opt to use a proxy to retrieve the information in order to protect the user from their IP address from being leaked.**

latex — Represent a LaTeX equation. Provide the LaTeX equation.

code — A block of code. Provide the programming language in question.

orlist — An ordered, roman-numeral list of things. Provide a JSON list!

onlist — An ordered—but with regular numbers—list of things. Provide a list.

uplist — An unordered list with dots (or what the client likes). Provide a list.

udlist — An unordered list with dashes. Provide a list.

ublist — An unordered list with boxes. Provide a list.

table — Specify a table. Provide a JSON object.

- **You must have a field called "fields", which is an array of the fields (e.g., "name", "age", "grade", etc). The index number of these will correspond downwards.**
- **Then, you must have a field called "entries" which is an array of arrays which correspond to the fields above.**
- **Example:**

```
{
    "fields": ["name", "age", "grade"],
    "entries": [
        ["John", 15, 9],
        ["David", 17, 11],
        …
    ]
}
```

would yield:

name	*age*	*grade*
John	**15**	**9**
David	**17**	**11**