**What’s the Dealio Protocol (WTDP)**

What’s the Dealio Protocol (WTDP) is a chatroom-based protocol that defaults on the port **8029.** WTDP sends all data through a JSON object known as a dealio. Each dealio has a type which is determined by what currently needs to be transferred and the data contained in that type. WTDP provides support for many optional features which includes: Direct-Messaging, current user list, and special content. See the below documentation on how to add or remove these features from your application.

Standard operation is as follows:

* **Client Connection** contains the [request](#_xbqy5b6e1ca9) sent by the client and the [response](#_aoco7g4c551n) sent back from the server. In joining a server, the client sends the server a requested username and in response the server returns an id not currently being used by another client and a list of currently connected users. The list of currently connected users may not be supported, if this is the case the list will contain the message “User list not supported.” If client does not implement the user list, do not do anything with the data. The server will also send an [update dealio](#_3c2n13k6jnk4) to all other users that this new client has connected and who it is.
* **Sending Messages** between users falls into two categories: broadcast or direct-message. Servers that do not implement direct-messaging can return an [error dealio](https://docs.google.com/document/d/1RFxAbGagYV8GvhyEEG_-RId2FlObrkw_ztooBHXWPIY/edit#heading=h.6p1oiqcybe7b) to the client. Sending messages is described in [this dealio](#_1tyt8lfr3cl2). If the server implements special content types such as pictures use this [special dealio](#_sixn3s7eoqkm). Otherwise it will return an [error dealio](https://docs.google.com/document/d/1RFxAbGagYV8GvhyEEG_-RId2FlObrkw_ztooBHXWPIY/edit#heading=h.6p1oiqcybe7b). If the requested message is OK, the server then sends a [broadcast dealio](#_uqqsdgmsc3aq) to all specified clients as well as the sender client which acts as a confirmation.
* **Leaving the Server** consists of 2 dealios which allow the other users to be informed of a client leaving the server. When a client wants to leave the server they can send a [end dealio](#_njbbhkrd1q8o). This is a courtesy dealio which helps speed up the server but the client can also leave the server without sending this dealio. If the dealio is received or the socket is closed the server will then identify the id of the user that left and then send an [update dealio](#_3c2n13k6jnk4) which tells all active users that a client has disconnected.
* **Returning error messages from the server** is covered by the [error dealio](https://docs.google.com/document/d/1RFxAbGagYV8GvhyEEG_-RId2FlObrkw_ztooBHXWPIY/edit#heading=h.6p1oiqcybe7b) which provides the client with an error in their request. See the dealio for a full list of error messages and their description.

## Application Level Decisions:

There are multiple application level decisions that needs to be made when implementing WTDP. WTDP was built to allow the maximum amount of developer freedom with as much standardization as possible. When developing applications using WTDP the following aspects need to be discussed in order to optimize your application to the protocol.

**For Server Applications:**

* Maximum number of users supported on your server. The server should return -1 as an id if the server is full and close the connection to the client attempting to connect.
* If your server supports a list of currently connected users it should send this in the “chatroom-response” dealio, otherwise it should send a not supported message.
* Special message content. If your server application allows the transfer of pictures/GIFs it should use the “chatroom-special” dealio, otherwise it should return the client a not supported error dealio.
* We advise implementation of a map between unique IDs currently in use and their username in order for proper transfer of messages.

**For Client Applications:**

* If your client application supports a list of users it will be contained in the “chatroom response” dealio. It will then need to be updated when “chatroom-update” is sent by the server. If it doesn't it should not regard the list sent by the server.
* Special message content. Should your client application not support pictures, it should be able to handle the special dealio sent by the server.

# Different Types of Dealios:

## “Chatroom-begin” (Client->Server) //writethread

Chatroom-begin is the initialization dealio that the client sends to the server. The client sends the server the type along with the client’s username and the length of the username in the dealio. If the Username is acceptable and matches the length of the username the server will respond to the client in a dealio with type: “Chatroom-response”. If the username does not match the server specified requirements, the server will respond with type: “Chatroom-error” in a dealio.

* Contents of dealio:
  + Username - lower case
  + len(username) → **20 character limit -** lower case

Format: lowercase

{ type: “chatroom-begin”, username: “(Username selected by the user)”, len: (length of username) }

## “Chatroom-response”(Server->Client)

Chatroom-response is the servers default dealio response sent to the initial “chatroom-begin” type sent by the client. After verifying the username provided by the client, it will select a unique id not currently being used by another user and return that to the client. If there are no available id’s (ie. the server is full) it will return -1. It sends the client the number of current connected users and an array of users currently in the chatroom. If the server does not support a server list the array will contain “User list not supported” otherwise the array will contain the list of all usernames/ids in the format (username:id).

* Contents of dealio:
  + id assigned by server (range 0-(n-1) or -1 if no available id)
  + number of users (n) specified at the Application Layer
  + users: array of users currently in chatroom

Format:

{type: “chat-response”, id: (id assigned), clientNo: (number of clients in chatroom), users: [(array of users)]}

## “Chatroom-send”(Client->Server)

Chatroom-send is the basic send dealio that clients send to the server in order to send messages in plaintext. The client will send their ID as the from field. In the To field it will contain an array of the users they are sending the message to, or [](an empty array) if the message is being sent to everyone. The message will be sent in the message field and the message length will be the length of that message. The server will then check the clients object and then return a Chatroom-broadcast dealio to the user as a response or a chatroom-error if the dealio contains an error.

* Contents of dealio:
  + from:user id
  + to: an array of 1+ userID to send message to - message is broadcasted if array if empty)
  + message
  + message Length (max: 280)
* Format:
  + {type: “chatroom-send”, from: (userID), to: [(list of recipient id’s)], message: (message sent by user), message-length: (length of message)}

## “Chatroom-special”(Client->Server)

Chatroom-special is the advanced send dealio that clients send to the server in order to send other types of media. The client will send their id as the from field. In the To field it will contain an array of the users they are sending the message to, or [](an empty array) if the message is being sent to everyone. The message will be sent in the as a bytearray and the message length will be the length of that bytearray. The server will then check the clients object and then return a Chatroom-broadcast dealio to the user as a response or a chatroom-error if the dealio contains an error or the server does not support special message types.

* Contents of dealio:
  + from: user id
  + to: an array of 1+ userid to send message to - message is broadcasted if array if empty)
  + message\_type:
    - Types include:
      * Text
      * Gif
      * Png
      * Jpeg
  + message: (bytearray)
  + message length (max: 280)
* format:
  + {type: “chatroom-send”, from: (userID), to: [(list of recipient id’s)], message\_type: (see message\_types above), message: (message sent by user), message-length: (length of message)}

## “Chatroom-broadcast”(Server->Multiple Clients) //UserThread

The chatroom-broadcast dealio is sent to clients from the server when the server has a message to serve the client. If the message is designated to be sent to everyone, the server will send this dealio to everyone with the below fields. The from field contains the username concatenated with the id of the person who sent the message. The message will be the message that the client is trying to send everyone and length will be the length of the message. **NOTE:** The server should also send this dealio to the client who initially sent this message as a confirmation that the message was sent to everyone.

* Contents:
  + from: “username:id”
  + to (an array of 1+ userid’s to send message to - message is broadcasted if array if empty)
  + message\_type: (default to text but can be other supported types)
  + message
  + message length (max: 280)
* Format:
  + {type: “chatroom-broadcast”, from: (“username:id”), to: [(list of recipient id’s)], message: (message sent by user), message\_length: (length of message)}

## “Chatroom-end”(Client->Server)

This dealio type is specific for a user sending a request to leave the chat server. This is a courtesy message and is not required for a user to terminate the connection. When this message type is sent with the id, the server will close the socket connection to the user and reclaim the id to be distributed to another user who enters the server. If the client terminates the socket connection without a “chatroom-end” message the server will reclaim the unique id.

* Contents:
  + id
    - Unique id of the client who wants to disconnect
* Format:
  + { type:”chatroom-end”, id: (Unique id of the user) }

## “Chatroom-error” (Server->Client)

This dealio is sent back to the client in the event the client sends anything unexpected. See the below types of errors for error types.

* Contents:
  + id: id of user
  + type of error = [unexpected\_dealio\_type, malformed\_dealio, client\_time\_out, message\_exceeded\_max\_length, id\_not\_found, *unsupported\_file\_type*, *file\_size\_exceeded, user\_name\_length\_exceeded, special\_unsupported*]
* Format:
  + {type: “chatroom-error”, type\_of\_error: (error type specified above), id: (id of user)}

## “Chatroom-update”(Server-> Multiple Clients)

The chatroom-update dealio is sent whenever a client connects to the server or leaves the server(either through a “chatroom-end” dealio or a socket termination). It is sent to every user still on the chat server in order to inform them that a user has either joined or left the server.

* Contents:
  + type\_of\_update: can take value of ‘enter’ or ‘leave’
  + id: id of person who has entered or left the chatroom
* Format:
  + {type: “chatroom-update”, type\_of\_update: (update type specified above), id: (id of user)}

Limit of message length: 280 characters (same as a tweet)

Limit of users supported: specified at applicati

on level → using stress testing

**User Stories:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Users:  Apple  Berry  Cherry  Amount of users server allows: 2 | {  “type”: “chatroom-begin”,  “username”: “apple”,  “len”: 5  } | {  “type”: “chatroom-begin”,  “username”: “berry”,  “len”: 5  } | {  “type”: “chatroom-begin”,  “username”: “cherry”,  “len”: 6  } | {  “type”: “chatroom-send”,  “from”: 0,  “to”: [ ],  “message”: “whats up”,  “message\_len”: 9  } |
|  | {  “type”: “chatroom-response”  “id”: 0  “clientNo”: 1  } | {  “type”: “chatroom-response”  “id”: 1  “clientNo”: 2  } | {  “type”: “chatroom-response”  “id”: -1  “clientNo”: 2  } | {  “type”: “chat-room-broadcast”,  “from” : 0,  “to” : [0,1],  “message” : “whats up”,  “message-len”: 9  }  Displays to all clients:  Apple:0  Whats up |
|  | {  “type” : “chatroom-end”,  “id” : 1  } |  |  |  |

**Doc of discussion notes:**

<https://docs.google.com/document/d/1gBpjA1L1LSd02pmOlmvEYGgZ2hU5lEhM8d8PJHjP1E4/edit?usp=sharing>