Fluid Chat

Complete Implementation & Configuration Guide — v6.1

This comprehensive guide details the configuration process for the Fluid Chat plugin across both Web and Mobile responsive platforms. Follow these schemas and workflow structures precisely to guarantee elite application performance and seamless S3 storage synchronization.

1. Database Architecture & Schema

To fully leverage all messaging capabilities within Fluid Chat (including reactions, typing indicators, and media sharing), create the following three specific data types within your Bubble database ecosystem.

DATA TYPE: `CONVERSATION`

- **title** (Type: `text`)
- **members_number** (Type: `number`) — *Optimizes performance by removing the need to dynamically execute a count lookup on Conversation_Members continuously.*

DATA TYPE: MESSAGES

Field Name	Field Type	Description / Scope
audio_duration	number	Stores recorded file length in seconds **Mobile Only**
created_by	User	Standard Bubble message creator reference.
creation_date	date	Standard timestamp field.
conversation_id	Conversation	Relational reference mapping to the target Chat Conversation.
file_names_list	text (List)	Stores clean string names of uploaded files.
files_list	text (List)	Array storing file download URLs.
images_list	image (List)	Array storing absolute image URLs.
is_edited	yes/no	Flag checking if the message text was altered post-delivery.
is_system_message	yes/no	Flag used to style standard system event updates natively.
message	text	Main message text body.
reactions	text	Serialized string mapping active user reaction emojis.
read_by	User (List)	Array tracking which participants have read the message.
response_message_id	text	Stores target Message UniqueID when replying to an item.

DATA TYPE: CONVERSATION_MEMBERS

- **conversation_id** (Type: Conversation)
- **member** (Type: User)
- **is_writing** (Type: yes/no)

2. Fluid Chat (Web Edition Configurations)

Map the properties in the Bubble UI Property Editor exactly as detailed in the technical tables below to properly bind your relational architecture to the messaging display layout engine.

CORE DATA BINDING PROPERTIES

Plugin Editor Field	Target Property / Expression Mapping
Message Text List	`Messages.message`
Message Id List	`Messages.UniqueID`
Message Creator Id List	`Messages.created_by.UniqueID`
Message Creator Names List	`Messages.created_by.Name`
Timestamp	`Messages.creation_date`

USERS & MENTIONS CONFIGURATIONS

Plugin Editor Field	Target Property / Expression Mapping
Current User Id	`Current User's Unique ID`
Avatar List	`Messages.created_by.Image` (or Profile Picture field)
Mention Names List	`Messages (grouped by created_by)'s created_by's Name`
Mention Ids List	`Messages (grouped by created_by)'s created_by's UniqueID`
Mention Avatars List	`Messages (grouped by created_by)'s created_by's Image`

MEDIA, ENGAGEMENT & INDICATORS

Plugin Editor Field	Target Property / Expression Mapping
Images List	`Messages (format as text) -> This Message's images_list (join with ,)` Delimiter: | |
Files List	`Messages (format as text) -> This Message's files_list (join with ,)` Delimiter: | |
Files Names List	`Messages (format as text) -> This Message's file_names_list (join with ,)` Delimiter: | |
Reactions String	`Messages.reactions`
Reply to ID List	`Messages.response_message_id`
Is Edited List	`Messages.is_edited`
Custom Menu Items	Configurable text array mapping long-press/right-click trigger menu structures.
Read By Count List	`Messages.read_by (count)`
Required Read Count	Integer checking target minimum count value before marking bubble as officially read.
Is System Message List	`Messages.is_system_message`
Show Typing Indicator	Boolean flag switching visibility profiles of active typing objects.
Typing Users Text	Binds the parsed typing notifications payload string.

3. WEB WORKFLOWS IMPLEMENTATION

Workflow: Sending Images, Files, and Audio via Feed Interaction

1. User executes standard web selector operations clicking media/document attachments.
2. User updates file items actively (can optionally dismiss or add further items inside draft stage).
3. User hits Send button, triggering the **`Message Sent`** element event.
4. **Action Routine:** Instantiate a `Create a New Thing` step target data type **Messages** mapping fields as follows:
   - *created_by* = `Current User`
   - *creation_date* = `Current Date/Time`
   - *conversation_id* = Current actively loaded `Conversation`
   - *file_names_list* = `FluidChat's Uploaded File Name`
   - *files_list* = `FluidChat's Uploaded File List` (stores non-image files and audio binaries)
   - *images_list* = `FluidChat's Uploaded Images List`
   - *is_edited* = `no` | *is_system_message* = `no`
   - *message* = `FluidChat's Draft Text`
   - *read_by* = `Current User` (add item to list)
   - *response_message_id* = `FluidChat's Draft Reply To ID`

Workflow: Message Reactions Engine

1. User executes a context-menu trigger (right-click on desktop or long-press on touch screens) on a bubble.
2. User clicks an emoji glyph from the popover, triggering the **`Reaction Updated`** event.
3. **Action Routine:** Trigger `Make Changes to Thing` pointing to the targeted Message item:
   - *reactions* = `FluidChat's Updated Reactions String`

Workflow: Processing User @Mentions

1. When the character string token @ is keyed into the draft input container, the companion context user array triggers open automatically.
2. Selecting specific users formats tokens inside the input string. Sending triggers **`User Mentioned`**.
3. **Downstream Routine:** Bind programmatic notification actions directly downstream of this trigger hook, referencing the automated array state string: `FluidChat's Mentioned Users Ids`.

Workflow: Delete and Edit Message Routines

1. Context menu item interaction fires the underlying structural event hook: **`Long Press Triggered`**.
2. **Branch A (Edit Mode):** Only when `FluidChat's Last Action is edit`:
   - Action: `Make Changes to Thing` -> target Message: *is_edited* = `yes`, *message* = `FluidChat's Draft Text`.
3. **Branch B (Delete Mode):** Only when `FluidChat's Last Action is delete`:
   - Action: `Delete Thing` -> target constraint message matching: `Message's UniqueID = Selected Message ID`.

Workflow: Real-Time Typing Notifications Broadcast

- **Event A:** Do every time condition is met -> When `FluidChat's Is Typing is yes`:
  - Action: `Make Changes to Thing` -> target `Conversation_Members` row constraint matching current conversation and user -> set *is_writing* = `yes`.
- **Event B:** Do every time condition is met -> When `FluidChat's Is Typing is no`:
  - Action: `Make Changes to Thing` -> target `Conversation_Members` row constraint matching current conversation and user -> set *is_writing* = `no`.
- **UI Field Integration:** Map your element's property field value `Typing Users Text` dynamically using the query string: `Search for Conversation_Members (constraints: conversation_id = Current, is_writing = yes)'s member's Name join with "," "is typing..."`

Workflow: Optimized Infinite Scroll Pagination (Lazy Loading Setup)

- **Static Legacy Routing:** Simply pull query list using data binding constraint `creation_date Descending Order = no`.
- **Performant Lazy Loading Routing:** Construct primary source data search parameters specifying: `creation_date Descending Order = yes`, appended with modifier constraint operator `:items until "message_limit"`, concluded with post-sort query operator constraint `:sorted by creation_date Descending Order = no`.
  - *Design Note:* Declare a page-level Number custom state named `message_limit` initializing with base configuration boundaries (e.g., 20, 30, or 40).
- When scroll bounds hit upper limits, the element automatically fires event hook: **`Load More Triggered`**.
- **Action Routine:** Trigger `Set State` mapping parameter: `message_limit = parent_container's message_limit + X` (X represents your matching incremental load stepping window constant).

4. Fluid Chat (Mobile Native Edition Configurations)

The Native Mobile framework handles asset references dynamically using high-speed native device storage. Ensure the following specific property values are configured inside your native mobile application element viewport wrappers.

MOBILE ARCHITECTURE DATA PROPERTIES MAPPING

Mobile Plugin Editor Field	Target Property / Expression Mapping
Current User ID	`Current User.UniqueID`
Message IDs	`Messages.UniqueID`
Messages List	`Messages.message`
Creation Dates List	`Messages.creation_date`
Creator IDs	`Messages.created_by.UniqueID`
Creator Names	`Messages.created_by.Name`
Images List	`Messages.images_list`
File List	`Messages.files_list`
Audio Duration List	`Messages.audio_duration`
Incoming Attachment	Map directly to your local page-level string custom state variable tracking local cache image URI data before executing pipeline uploads.

5. Mobile Workflows & Local Cache Pipelines

Workflow: High-Speed Gallery Media Previews & Execution

1. User taps the Paperclip interface button component, executing the **`Attachment Pressed`** native container trigger event.
2. **Action 1:** Run Bubble Native Action **`Open camera library`**. Config requirements: Set Type configuration value profile to `Multiple Photos`; ensure `Optimize Image size` configuration parameter is actively **Checked/Ticked**. **Crucial for 100ms UI rendering**
3. **Action 2:** Run `Set State` updating your local page attachment state string tracking variable: `Attachment state = Result of Step 1 (Open Camera Library)'s each item's URL joined with "|||"`. *(Allows instant native multi-image preview render in the canvas layout without network overhead).*
4. User taps the primary Send action icon button, firing element event: **`Send Pressed`**.
5. **Action 3:** Run `Create a New Thing` under targeted table **Messages**:
   - *created_by* = `Current User` | *creation_date* = `Current Date` | *conversation_id* = `Conversation`
   - *images_list* = `FluidChat's Uploaded Files URLs`
   - *message* = `FluidChat's Typed Text` | *is_edited/is_system_message* = `no`
   - *read_by* = `Current User` | *response_message_id* = `FluidChat's Draft Reply To ID`

Workflow: High-Speed Real-Time Native Camera Processing

1. User taps the Camera icon button component, firing event: **`Camera Pressed`**.
2. **Action 1:** Run Bubble Native Action **`Open camera`**. Requirement: Ensure `Optimize Image size` configuration parameter is actively **Checked/Ticked**.
3. **Action 2:** Run `Set State` pointing to your page custom attachment tracker state variable: `Attachment state = Result of Step 1 (Open Camera)'s URL`.
4. User clicks Send, firing standard **`Send Pressed`** mobile event context layout.
5. **Action 3:** Execute `Create a New Thing` under targeted data table **Messages** mapping fields identical to the Gallery Media pipeline structure explained above.

Workflow: Asynchronous Voice Note Capture & Binary S3 Pipeline Upload

1. User performs tap gesture interaction over microphone container asset component, triggering native event: **`Mic Pressed`**.
2. **Action 1:** Invoke plugin context action module: **`Start Recording`**.
3. User taps native Send action vector button container inside operational audio tracking view state layout interface, triggering target event: **`ready_to_upload`** (Send Audio).
4. **Action 2:** Invoke operational pipeline plugin script action: **`Upload to Bubble`**. Set the dynamic string parameter property `Endpoint` precisely matching: `https://your-app.bubbleapps.io/version-test/fileupload`.
5. The hardware audio binary uploads to Amazon S3 in the background. On total upload success, the engine fires event trigger hook: **`Voice Uploaded Finished`**.
6. **Action 3:** Instantiate final structural data mapping record step via `Create a New Thing` targeting **Messages** schema fields:
   - *created_by* = `Current User` | *creation_date* = `Current Date` | *conversation_id* = `Conversation`
   - *files_list* = `FluidChat's Uploaded Voice URL split by "|||"`
   - *audio_duration* = `FluidChat's recording_duration`
   - *is_edited* = `no` | *is_system_message* = `no` | *read_by* = `Current User`
7. *Cancellation Handling:* If the user actively taps the trash container asset layer during live tracking capture modes, event hook **`Cancel Mic Pressed`** fires. Simply bind plugin element utility action wrapper **`Stop Recording`** directly beneath this event hook loop to clean local file cache paths automatically.