Graphics Programmer's Guide

Version 1.1.0

Component Layouts and Templating

Template system general overview

Template input data is used within the graphics template editor to populate dynamic content using a templating language called Handlebars.

Template input data

Each graphic, depending on the category of meeting data that it consumes and the data types contained within each category, will have distinct visual elements that will represent each of those data types. The following is a list of the meeting data categories that the Zoom Graphics Toolkit currently supports and the graphic types that consume them:

- **Meeting Chat Data**: Chat Scroll Graphic, Chat Highlight Graphic
- **Meeting QA Data**: QA Scroll Graphic, QA Highlight Graphic
- **Meeting Poll Data**: Poll Result Graphic
- **Meeting Emoji Reaction Data**: Emoji Reaction Graphic
- **Meeting Caption Data**: Closed Caption Graphic
- **Meeting Engagement Counts**: Counter Graphic
- **Meeting Participant Data**: Participant List Graphic

Two graphic types are special cases. The Counter Graphic does not consume a single data category; instead it aggregates live *counts* drawn from several of the categories above (chat, QA, polls, and emoji reactions) along with participant statistics. The Combiner Graphic consumes no meeting data at all — it composes other graphics and images into a single layered output.

As stated above, the Chat Scroll Graphic consumes the Meeting Chat Data category. Composed in the meeting chat data is a list of `**chat_message**` and `**chat_reply**` objects. These are similar but distinct data types with different attributes. A `**chat_reply**` object in this case has all the properties of a `**chat_message**` object and an additional property called "parentMessage". The value of the "parentMessage" property is a reference to the chat message that started the thread that the chat reply belongs to.

All meeting data are JSON-serializable JavaScript objects under the hood. Therefore, we can define the data types of each graphic type more formally using the TypeScript syntax. The type definitions for each graphic type are as follows:

- Chat scroll graphic template input
- Chat highlight graphic template input
- Closed captions graphic
- Counter graphic
- Emoji reaction graphic
- Participant list graphic
- Poll result graphic
- QA highlight graphic
- QA scroll graphic

Chat scroll graphic template input

chat_message

```
1 type ChatMessageData = {
```

```
  messageID: string
  threadID: string
  timestamp: number
  chatMessageType: number
  senderUserID: string
  senderDisplayName: string
  senderAvatarPath: string | undefined
  msgContent: string
  isComment: boolean
  isChatToWaitingRoom: boolean
  isThread: boolean
  isEmphasized: boolean
  textStyleItemList: any[]
}
```

chat_reply

```
type ChatReplyData = {
  messageID: string
  threadID: string
  timestamp: number
  chatMessageType: number
  senderUserID: string
  senderDisplayName: string
  senderAvatarPath: string | undefined
  msgContent: string
  isComment: boolean
  isChatToWaitingRoom: boolean
  isThread: boolean
  isEmphasized: boolean
  textStyleItemList: any[]
  parentMessage:
    | {
        messageID: string
        threadID: string
        timestamp: number
        chatMessageType: number
        senderUserID: string
        senderDisplayName: string
        senderAvatarPath?: string
        msgContent: string
        isComment: boolean
        isChatToWaitingRoom: boolean
        isThread: boolean
        isEmphasized: boolean
        textStyleItemList: any[]
      }
    | undefined
```

```
}
```

Chat highlight graphic template input

chat_message

```
type ChatMessageData = {
  messageID: string
  threadID: string
  timestamp: number
  chatMessageType: number
  senderUserID: string
  senderDisplayName: string
  senderAvatarPath: string | undefined
  msgContent: string
  isComment: boolean
  isChatToWaitingRoom: boolean
  isThread: boolean
  isEmphasized: boolean
  textStyleItemList: any[]
}
```

chat_reply

```
type ChatReplyData = {
  messageID: string
  threadID: string
  timestamp: number
  chatMessageType: number
  senderUserID: string
  senderDisplayName: string
  senderAvatarPath: string | undefined
  msgContent: string
  isComment: boolean
  isChatToWaitingRoom: boolean
  isThread: boolean
  isEmphasized: boolean
  textStyleItemList: any[]
  parentMessage:
    | {
        messageID: string
        threadID: string
        timestamp: number
        chatMessageType: number
        senderUserID: string
```

```
        senderDisplayName: string
        senderAvatarPath: string | undefined
        msgContent: string
        isComment: boolean
        isChatToWaitingRoom: boolean
        isThread: boolean
        isEmphasized: boolean
        textStyleItemList: any[]
      }
    | undefined
}
```

Closed captions graphic

single_language_caption

```
type singleLanguageCaptions = {
  messageID: string
  speakerID: number
  speakerName: string
  messageContent: string
  timeStamp: number
  messageType: number
  senderAvatarPath?: string
  isOriginal: boolean
  isSim?: boolean
  strMarker?: number
}[]
```

multi_language_caption

```
type multiLanguageCaptions = {
  original: {
    messageID: string
    speakerID: number
    speakerName: string
    messageContent: string
    timeStamp: number
    messageType: number
    senderAvatarPath?: string
    isOriginal: boolean
    isSim?: boolean
    strMarker?: number
  }
  translation?: {
```

```
    messageID: string
    speakerID: number
    speakerName: string
    messageContent: string
    timeStamp: number
    messageType: number
    senderAvatarPath?: string
    isOriginal: boolean
    isSim?: boolean
    strMarker?: number
  }
}[]
```

traditional_captions

```
type traditionalCaptions = {
  messageID: string
  speakerID: number
  speakerName: string
  messageContent: string
  timeStamp: number
  messageType: number
  senderAvatarPath?: string
  isOriginal: boolean
  isSim?: boolean
  strMarker?: number
}
```

Counter graphic

The Counter graphic's HTML template is rendered once per visible counter. Each render receives the following context:

```
type CounterTemplateContext = {
  counterKey: string // identifier of the counter, e.g. 'total_chat_messages'
  icon: string // the configured icon identifier or emoji
  iconColor: string // resolved icon color ('' if unset)
  iconHtml: string
// pre-rendered HTML for the icon; use with a triple-stache: {{{iconHtml}}}
  value: string
// the formatted value (numbers are localized; elapsed time is h:mm:ss / m:ss)
  label: string // the configured label text
  anchorGroups: {
```

```
    h: string
  // horizontal anchor shared by this group's elements, e.g. 'left', 'center',
  'right'
    v: string
  // vertical anchor shared by this group's elements, e.g. 'top', 'center',
  'bottom'
    items: { type: 'icon' | 'value' | 'label' }[]
  // the visible elements anchored here, in display order
  }[]
}
```

The icon, value, and label are grouped by which corner/edge of the counter card they're anchored to (set per-element in the Graphic Designer's Counter Content Order and per-element alignment controls). Elements that share the same horizontal and vertical anchor are grouped together so they can be laid out as a unit.

The underlying counter values are aggregated into a **`CounterData`** object on the backend, keyed by counter identifier:

```
type CounterData = {
  [counterKey: string]: number | string | undefined
}
```

The default template iterates **`anchorGroups`**, and within each group iterates **`items`**, to place the icon, value, and label in the order and grouping configured in the Graphic Designer:

```
{{#each anchorGroups}}
<div class='counter-anchor counter-anchor-{{h}}-{{v}}'>
{{#each items}}
{{#if (Eq type "icon")}}<span class='counter-icon'>{{{../../iconHtml}}}</span>{{
/if}}
{{#if (Eq type "value")}}<span class='counter-value'>{{../../value}}</span>{{/if
}}
{{#if (Eq type "label")}}<span class='counter-label'>{{../../label}}</span>{{/if
}}
{{/each}}
</div>
{{/each}}
```

Emoji reaction graphic

The emoji graphic currently does not expose any meeting data to the templates.

Participant list graphic

The Participant List graphic uses two separate HTML templates: one rendered once per participant row, and one rendered once per group header (used by list types that group participants, such as Grouped by Role or Yes / No Voting). Ungrouped list types render only participant rows.

The participant row template receives the following context:

```
type ParticipantRowContext = {
  number: number // the participant's position in the list, starting at 1
  showNumber: boolean // whether Row Numbering is enabled in List Customization
  showAvatar: boolean
// whether Profile Image is set to anything other than 'None'
  avatarUrl: string
// the participant's photo URL; '' if unavailable or Profile Image is not 'Real
Photo'
  initials: string
// computed from the participant's name; used as the avatar fallback
  name: string
// formatted per the Name Style setting (Full Name / Initials / First Initial +
Last Name)
  statusIconHtml: string
// pre-rendered HTML for the row's status icon (e.g. an emoji reaction or vote
icon); use with a triple-stache: {{{statusIconHtml}}}
  detailText: string // reserved for future use; currently always ''
}
```

The group header template receives:

```
type GroupHeaderContext = {
  groupLabel: string // e.g. 'Hosts', 'Muted', 'Yes'
  count: number // number of participants in this group
}
```

Both templates are rendered against a **`ParticipantRecord`** for each participant, aggregated on the backend into a **`ParticipantListData`** object:

```
type ParticipantRecord = {
  userID: number
  name: string
  role: 'host' | 'cohost' | 'panelist' | 'attendee'
  isVideoOn: boolean
  isAudioMuted: boolean
  isRaiseHand: boolean
  raiseHandOrder: number | null
  joinOrder: number
  isInWaitingRoom: boolean
  avatarUrl: string | null
  reaction: 'clap' | 'thumb' | 'heart' | 'joy' | 'openmouth' | 'tada' | null
  reactionOrder: number | null
  voteState: 'yes' | 'no' | 'not_voted'
}

type ParticipantListData = {
  participants: ParticipantRecord[]
  timestamp: number
  isWebinar?: boolean
  // true when the current session is a webinar, which affects role labels
}
```

The default templates render as follows:

```
{{! participant row }}
{{#if showNumber}}<span class='pl-number'>{{number}}</span>{{/if}}
{{#if showAvatar}}
  <span class='pl-avatar'>
    {{#if avatarUrl}}<img class='pl-avatar-img' src='{{avatarUrl}}' alt='' />{{
else}}<span
        class='pl-avatar-initials'
      >{{initials}}</span>{{/if}}
  </span>
{{/if}}
<span class='pl-name'>{{name}}</span>
{{#if statusIconHtml}}<span class='pl-status-icon'>{{{statusIconHtml}}}</span>{{
/if}}
```

```
{{! group header }}
<span class='pl-group-label'>{{groupLabel}}</span>
<span class='pl-group-count'>{{count}}</span>
```

Poll result graphic

question_title

```
type PollQuestionItemData = {
  pollingID: string
  pollingQuestionID: string
  pollingQuestionName: string
  pollingQuestionType: 1 /* SingleSelect */ | 2 /* MultiSelect */
  answeredCount: number
  isRequired: boolean
  pollingSubQuestionItemList: {
    pollingParentQuestionID: string
    pollingSubQuestionID: string
    pollingQuestionName: string
  }[]
  questionImagePath: string | undefined
  visMethod: 'pie-chart' | 'bar-chart'
  graphicStatus: { showingResults: boolean; showingQuestion: boolean }
  result: {
    totalResponseCount: number | undefined
    questionResults:
      | {
          [answerID: string]: {
            answerID: string
            answerValue: string | undefined
            frequency: number
            isCorrect: boolean | undefined
          }
        }
      | undefined
    subQuestionResults:
      | {
          [pollingSubQuestionID: string]: {
            [answerID: string]: {
              answerID: string
              answerValue: string | undefined
              frequency: number
              isCorrect: boolean | undefined
            }
          }
        }
      | undefined
  }
}
```

bar_chart

```
type PollQuestionItemData = {
  pollingID: string
```

```
  pollingQuestionID: string
  pollingQuestionName: string
  pollingQuestionType: 1 /* SingleSelect */ | 2 /* MultiSelect */
  answeredCount: number
  isRequired: boolean
  pollingSubQuestionItemList: {
    pollingParentQuestionID: string
    pollingSubQuestionID: string
    pollingQuestionName: string
  }[]
  questionImagePath: string | undefined
  visMethod: 'pie-chart' | 'bar-chart'
  graphicStatus: { showingResults: boolean; showingQuestion: boolean }
  result: {
    totalResponseCount: number | undefined
    questionResults:
      | {
          [answerID: string]: {
            answerID: string
            answerValue: string | undefined
            frequency: number
            isCorrect: boolean | undefined
          }
        }
      | undefined
    subQuestionResults:
      | {
          [pollingSubQuestionID: string]: {
            [answerID: string]: {
              answerID: string
              answerValue: string | undefined
              frequency: number
              isCorrect: boolean | undefined
            }
          }
        }
      | undefined
  }
}
```

pie_chart

```
type PollQuestionItemData = {
  pollingID: string
  pollingQuestionID: string
  pollingQuestionName: string
  pollingQuestionType: 1 /* SingleSelect */ | 2 /* MultiSelect */
  answeredCount: number
  isRequired: boolean
```

```
  pollingSubQuestionItemList: {
    pollingParentQuestionID: string
    pollingSubQuestionID: string
    pollingQuestionName: string
  }[]
  questionImagePath: string | undefined
  visMethod: 'pie-chart' | 'bar-chart'
  graphicStatus: { showingResults: boolean; showingQuestion: boolean }
  result: {
    totalResponseCount: number | undefined
    questionResults:
      | {
          [answerID: string]: {
            answerID: string
            answerValue: string | undefined
            frequency: number
            isCorrect: boolean | undefined
          }
        }
      | undefined
    subQuestionResults:
      | {
          [pollingSubQuestionID: string]: {
            [answerID: string]: {
              answerID: string
              answerValue: string | undefined
              frequency: number
              isCorrect: boolean | undefined
            }
          }
        }
      | undefined
  }
}
```

QA highlight graphic

question

```
type QuestionData = {
  questionID: string
  timestamp: number
  senderName: string | undefined
  questionContent: string
  isAnonymous: boolean
  upvoteNum: number
  liveAnsweringUserName: string | undefined
```

```
  isBeingLiveAnswered: boolean
  isMarkedAsAnswered: boolean
}
```

answer

```
type AnswerData = {
  questionID: string
  answerID: string
  timestamp: number
  answerContent: string
  senderName: string | undefined
  isLiveAnswer: boolean
}
```

QA scroll graphic

question

```
type QuestionData = {
  questionID: string
  timestamp: number
  senderName: string | undefined
  questionContent: string
  isAnonymous: boolean
  upvoteNum: number
  liveAnsweringUserName: string | undefined
  isBeingLiveAnswered: boolean
  isMarkedAsAnswered: boolean
}
```

answer

```
type AnswerData = {
  questionID: string
  answerID: string
  timestamp: number
  answerContent: string
  senderName: string | undefined
  isLiveAnswer: boolean
}
```

Active Speaker Name Tag

name_tag

```
<div class='name-tag-container-inner'>
  <div class='display-name-wrapper'>
    <span class='display-name'>{{Titlecase activeSpeakerName}}</span>
  </div>
</div>
<div class='backdrop'></div>
```

Chat scroll

chat_message example

```
<div class='template-chat-message-container'>
  {{#if senderAvatarPath}}
    <img class='sender-avatar' src='{{senderAvatarPath}}' />
  {{/if}}
  <div>
    <div>
      <span class='sender-display-name'>{{Titlecase (
ShortenLastName senderDisplayName)}}</span>
      <time class='chat-timestamp'>{{FormattedTime timestamp 'p'}}</time>
    </div>
    <p class='msg-content'>{{msgContent}}</p>
  </div>
</div>
```

Chat highlight

chat_reply example

```
<div class='template-chat-comment-container'>
  <div class='parent-thread-reference-container'>
    <span class='replying-to-info'>
      Replying to thread by
      <span class='replying-to-display-name'>{{parentMessage.senderDisplayName}}
</span>
    </span>
    <p class='parent-thread-msg-qoute'>{{parentMessage.msgContent}}</p>
```

```
  </div>
  <div class='comment-msg-container'>
    {{#if senderAvatarPath}}
      <img class='sender-avatar' src='{{senderAvatarPath}}' />
    {{/if}}
    <div>
      <div>
        <span class='sender-display-name'>{{Titlecase (
ShortenLastName senderDisplayName)}}</span>
        <time class='chat-timestamp'>{{FormattedTime timestamp 'p'}}</time>
      </div>
      <p class='msg-content'>{{msgContent}}</p>
    </div>
  </div>
</div>
```

Closed Captions

caption_box

```
{{#each singleLanguageCaptions}}
  <div>
    <div>{{FormattedTime this.timeStamp 'p'}}</div>
    <div>Speaker name: {{this.speakerName}}</div>
    <div>Caption message content: {{this.messageContent}}</div>
  </div>
  <br />
{{/each}}
```

Emoji Reactions

fountain_box

```
<div class='emoji-page'>
  <div class='row emoji-box' id='emoji-box'>
    <div class='wrapper'>
      <div id='spawn-clap' class='clap emoji-icon'></div>
    </div>
    <div class='wrapper'>
      <div id='spawn-thumb' class='thumb emoji-icon'></div>
    </div>
    <div class='wrapper'>
      <div id='spawn-heart' class='heart emoji-icon'></div>
```

```
    </div>
    <div class='wrapper'>
      <div id='spawn-joy' class='joy emoji-icon'></div>
    </div>
    <div class='wrapper'>
      <div id='spawn-openmouth' class='openmouth emoji-icon'></div>
    </div>
    <div class='wrapper'>
      <div id='spawn-tada' class='tada emoji-icon'></div>
    </div>
  </div>
</div>
```

Poll result

question_title

```
<div class='poll-question-title-container'>
  <div class='title'>{{pollingQuestionName}}</div>
</div>
```

bar_chart

```
<div class="bar-chart-container">
  <div class="category-label-list-container">
    {{#each result.questionResults}}
    <div class="category-label">{{this.answerValue}}</div>
    {{/each}}
  </div>
  <div
    class="bar-list-container"
    style="--answer-item-count:{{Measurable-GetLength result.questionResults}}"
  >
    {{#each result.questionResults}}
    <div
      class="bar"
      style="--scale: calc(100%*({{this.frequency}})/max({{#each
../result.questionResults}}{{this.frequency}}, {{/each}}0));"
    ></div>
    {{/each}}
  </div>
</div>
```

pie_chart

```
<div class="pie-chart-container">
  <div class="pie-chat-result-container">
    {{#each result.questionResults}}
      <div class="pie-chart-slice" style="...">
        {{#if this.frequency}}
        <svg viewBox="0 0 200 200" style="...">
          <circle
            cx="100"
            cy="100"
            r="50"
            fill="none"
            stroke-width="100px"
            stroke-dasharray="314"
            stroke-dashoffset="calc(314 * (1 - {{this.frequency}}/
{{../result.totalResponseCount}}))"
          />
        </svg>
        {{/if}}
      </div>
    {{/each}}
    {{#each result.questionResults}}
      <div class="percentage-label" style="...">
        <span style="...">{{Math-Multiply (
Math-Divide this.frequency ../result.totalResponseCount 6) '100' 1}}%
      </div>
    {{/each}}
    </div>
  </div>
</div>
```

QA Highlight

answer example

```
<div class='template-answer-container'>
  {{#if senderAvatarPath}}
    <img class='sender-avatar' src='{{senderAvatarPath}}' />
  {{/if}}
  <div class='answer-text-container'>
    <p>Sender name: {{senderName}}</p>
    <p>Answer content: {{answerContent}}</p>
    <p>Timestamp: {{timestamp}}</p>
    <p>isLiveAnswer: {{isLiveAnswer}}</p>
  </div>
</div>
```

QA Scroll

question example

```
<div class='template-question-container'>
  <div>
    <div>
      {{#if senderName}}
        <span class='sender-display-name'>{{Titlecase senderName}}</span>
      {{else}}
        <span class='sender-display-name'>Anonymous</span>
      {{/if}}
      <time class='question-timestamp'>{{FormattedTime timestamp 'p'}}</time>
    </div>
    <p class='question-content'>{{questionContent}}</p>
  </div>
</div>
```

Handlebars

Overview

Meeting data is consumed using Handlebars. Handlebars provide built-in helpers that are used in the graphic templates. By leveraging these built-in Handlebars helpers, graphic templates can efficiently process and present meeting data in a variety of ways to meet different visualization needs.

Details of the Handlerbars templating language documentation can be found here

Custom helpers

The Zoom Graphic Toolkit provides defined helper functions to manipulate data. These helper functions include string manipulation, math operations, and other utility functions such as formatting timestamps.

ConvertNameToInitials

Convert the name into its first and last initials

```
function (text)
```

text - string
return - string

```
{{ConvertNameToInitials 'John Doe'}}

// Results: JD
```

Eq

Check if two arguments are equal

```
function (arg1, arg2)
```

arg1 - First argument of type any
arg2 - Second argument of type any
return - boolean

```
{{Eq 'option_1' 'option_2'}}

// Results: false
```

FormattedTime

Format the epoch timestamp to a long localized time in the format of HH:MM AM/PM using the pattern “p”
FormattedTime is a wrapper of the format() method from the date-fns library. The user can specify the format of the timestamp based on the formatStr pattern passed in. Refer to the date-fns library for a list of format patterns.

```
function (timestampStr, formatStr)
```

timestampStr - string (epoch time)
formatStr - string (a pattern from the date-fns library)

return - string

```
{{FormattedTime '1744399255000' 'p'}}
// Results: "12:20 PM"

{{FormattedTime '1744399255000' 'pp'}}
// Results: "12:20:55 PM"
```

Math-Add

Return the sum of two numbers while specifying the number of decimals to round

```
function (numAStr, numBStr, roundToDecimal)
```

numAStr - string
numBStr - string
roundToDecimal - number
return - string

```
{{Math-Add '4' '1' 0}}

// Results: "5"
```

Math-Compare

Compare if numAstr is greater than numBstr

```
function (numAStr, numBStr)
```

numAStr - string
numBStr - string
return - boolean

```
{{Math-Compare '4' '1'}}

// Results: true
```

Math-Cos

Return the cosine of a number while specifying the number of decimals to round

```
function (numStr, roundToDecimal)
```

numAStr - string (in radians)
roundToDecimal - number
return - string

```
{{Math-Cos '.4089' 4}}

// Results: "0.9176"
```

Math-Divide

Divide two numbers while specifying the number of decimals to round

```
function (numAStr, numBStr, roundToDecimal)
```

numAStr - string
numBStr - string
roundToDecimal - number
return - string

```
{{Math-Divide '7' '3' 1}}

// Results: "2.3"
```

Math-Multiply

Multiply two numbers while specifying the number of decimals to round

```
function (numAStr, numBStr, roundToDecimal)
```

numAStr - string
numBStr - string
roundToDecimal - number
return - string

```
{{Math-Multiply '10' '.75' 1}}
// Results: 7.5
```

Math-Sin

Return the sine of a number while specifying the number of decimals to round

```
function (numStr, roundToDecimal)
```

numAStr - string (in radians)
roundToDecimal - number
return - string

```
{{Math-Sin '.4089' 4}}

// Results: "0.3976"
```

Math-Subtract

Return the difference of two numbers while specifying the number of decimals to round

```
function (numAStr, numBStr, roundToDecimal)
```

numAStr - string
numBStr - string
roundToDecimal - number
return - string

```
{{Math-Subtract '4' '1' 0}}

// Results: "3"
```

Measurable-GetLength

Get the length of a string or the number of keys in an object or the length of an array

```
function (obj)
```

obj - object or array or string
return - string

SelectPseudoRandomly

Randomly select an item from the passed in arguments

```
function (...args)
```

args - All of the arguments passed in using the handler bar helper (see example). The first argument is a string that is used to generate a random seed. The random seed is then used to randomly select the rest of the arguments
return - string

```
{{SelectPseudoRandomly 'TestSeed' 'foo' 'bar' 'baz'}}

// Result: "bar"
```

ShortenLastName

Shorten the string of names to only the first full name and last name initial

```
function (text)
```

text - string
return - string

String-Split

Split a string into substrings using the specified separator and return them as an array

```
function (str, separator)
```

str - string
separator - string. It identifies character or characters to use in separating the string
return - array of strings

```
{{String-Split 'Hello World' ' '}}

// Results: ["Hello", "World"]
```

Titlecase

Uppercase the beginning of every word of the string

```
function (text)
```

text - string
return - string

Framer Motion Presence Animation CSS Hooks

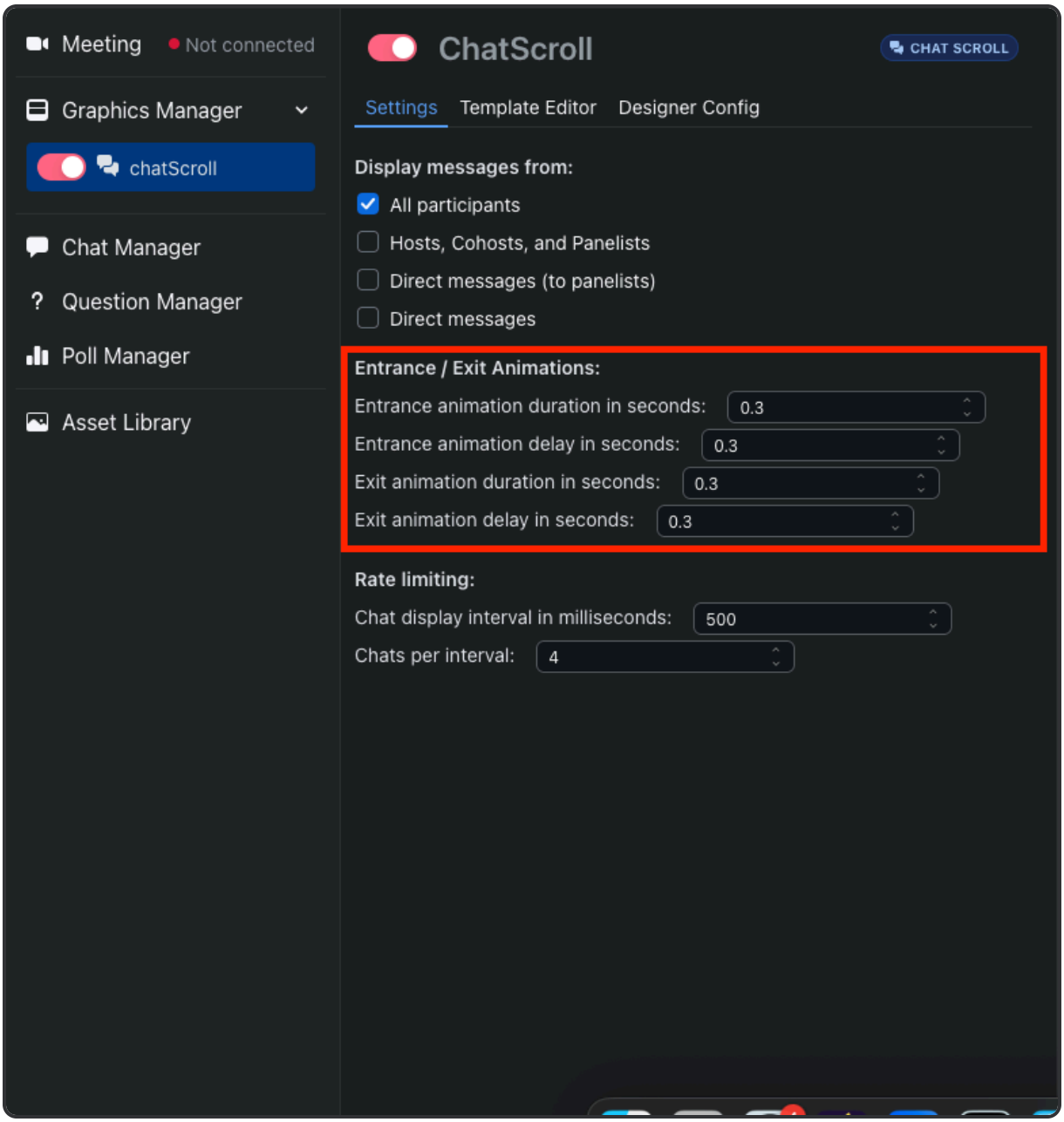
The Zoom Graphics Toolkit uses Framer Motion's **AnimatePresence** to animate components in and out of view across different graphic types.

This means that every time a graphic is:

- **Rendered** → an **entrance animation** runs.
- **Removed** → an **exit animation** runs.

Timing Configuration and Constraints

- You can customize the entrance/exit timing in the graphic designer's settings panel:



- Duration constraints: 0.0 seconds to 10.0 seconds
- Delay constraints: 0.0 seconds to 10.0 seconds

Parent Containers

All HTML written by the user in the *Template Editor* tab exists within a parent div of:

- **chat-message-container** or **chat-reply-container** (for Chat Scroll and Chat Highlight graphics)

- **answer-message-container** or **question-message-container** (for QA Scroll and QA Highlight graphics)

Those divs also exist within a parent div of **content-area** (regardless of graphic type). It is important to implement styling in the CSS for the automatically injected divs in order for animations to apply.

The following is an example of a chat highlight graphic that manipulates these parent containers.

```
<div class='chat-highlight-lower-third'>
  {{#if senderAvatarPath}}
    <img class='chat-avatar' src='{{senderAvatarPath}}' alt='Avatar' />
  {{/if}}
  <div class='chat-content'>
    <div class='chat-header'>
      <span class='sender-name'>{{Titlecase senderDisplayName}}</span>
      <time class='chat-time'>{{FormattedTime timestamp 'p'}}</time>
    </div>
    <p class='chat-message'>{{msgContent}}</p>
  </div>
</div>
```

```
.content-area {
  height: 100%;
}

.chat-message-container {
  position: relative;
  height: 100%;
  width: 100%;
}

.chat-reply-container {
  position: relative;
  height: 100%;
  width: 100%;
}

.chat-highlight-lower-third {
  position: absolute;
  bottom: 10%;
  left: 50%;
  transform: translateX(-50%);
  background-color: var(--background-color, #000000aa);
  color: var(--text-color, #ffffff);
```

```
  padding: var(--padding, 10px);
  border-radius: var(--border-radius, 5px);
  font-family: var(--font-family, 'Almaden Sans'), sans-serif;
  font-size: var(--font-size, 14px);
  display: flex;
  align-items: center;
  max-width: 80%;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.3);
}
```

Note that in the above example, it is critical that the CSS addresses style for **`.content-area`**, **`.chat-message-container`** and **`.chat-reply-container`** in order for the Motion animations to apply.

CSS Hooks

Users can add additional animation styling using the *framer-animate-presence-exiting* class which is applied to an element that is about to be removed.

The following is an example of how a poll graphic type might use this class.

```
.question-title-container.framer-animate-presence-exiting {
  animation-name: slide-out;
  animation-delay: 0s;
  animation-duration: 0.5s;
  animation-timing-function: ease-in;
  animation-fill-mode: both;
  animation-direction: normal;
}
```

Here's another example, this time with a chat scroll graphic type with a custom animation.

```
.template-chat-message-container.framer-animate-presence-exiting {
  animation-name: fade-slide-out;
  animation-duration: 0.5s;
  animation-fill-mode: both;
  animation-timing-function: ease-in;
}

@keyframes fade-slide-out {
  0% {
```

```
    transform: translateX(0);
    opacity: 1;
  }
  100% {
    transform: translateX(100%);
    opacity: 0;
  }
}
```

Syntax

Graphic designer form generation

Syntax

Since the graphic designer form configuration is represented as a JSON format, users must adhere to JSON syntax rules.

- keys and values must be enclosed with double quotes NOT single quotes
- The version number must be a semantic version number. An example would be “1.0.0”. (see https://semver.org/ for more information).

```
// This is correct ✅
"version": "1.0.0"
"form-checkbox": "true"
"max": "40"

// This is wrong 🚫
"version": 1.0.0
"version": "v1.0"
version: "1.0.0"
'version': "1.0.0"
'version': '1.0.0'
"form-checkbox": true
"max": 40
```

- JSON data must be followed by a comma when there is more data after it, otherwise you will encounter an error that prevents saving your changes

```
// This is correct ✅
{
   "version": "0.0.1",
   "form": [...]
}

// This is correct ✅
{
   "version": "0.0.1",
   "form": [...],
   "sectionMacros": {...}
}

// This is wrong 🚫
{
   "version": "0.0.1",
   "form": [...],
// There is no data after "form" so this comma will cause an error
}
```

- Escape the quotation character "" using the backslash \ if you want to include it part of your string

```
// This is correct ✅
```

```
"css-image": "url(\"\")"

// This is wrong 🚫
"css-image": "url("")"
```

Configuration property references

- The code snippets and tables below serve as references for the properties that can be utilized in the designer form configuration
- Form vs Macro
  - **Form**: The general layout of what the design form will include
  - **Macro**: The pre-programmed pieces of the design form that allows you to adjust the designs of your graphic. With these macros, you can easily update colors, sizes, layouts, and other design properties without rebuilding the entire graphic from scratch.
    - The benefit of setting parameterMacros and sectionMacros is that they are re-usable. The macros can be used multiple times without having to re-create a GraphicDesignerFormSection object or GraphicDesignerParameter inside the form

Design form configuration

Typescript definition

```
type GraphicDesignerFormConfig = {
  version: string
  parameterMacros?: { [macroName: string]: GraphicDesignerParameter } // optional
  sectionMacros?: { [macroName: string]: GraphicDesignerFormSection } // optional
  form: (GraphicDesignerFormSection | SectionMacroReference)[]
}
```

JSON example

```
{
  "version": "1.0.0",
  "form": ["BACKGROUND"]
}
```

Table reference

Property	Type	Required
version	string	
parameterMacros	`{ [ macroName: string ]: GraphicDesignerParameter }`	optional
sectionMacros	`{ [ macroName: string ]: GraphicDesignerFormSection }`	optional
form	Array of GraphicDesignerFormSection and SectionMacroReference	

Default macro sections

The following list are names (macroName) for default macro presets that the user can choose from. Each of the presets follows the syntax of the GraphicDesignerFormSection object.

- BOUNDS

```
{
  "label": "Bounds",
  "slug": "bounds",
  "subSectionsPresentation": "column",
  "subsections": [
    {
      "slug": "vertical",
      "parametersPresentation": "row",
      "parameters": [
        {
          "label": "Vertical Anchor",
          "slug": "anchor",
          "types": ["gfx-global-variable", "form-select"],
          "defaults": {
            "form-select": "top"
          },
          "defaultType": "form-select",
          "parameterTypeBasedOptions": {
            "form-select": {
              "options": [
                {
                  "label": "Top & Bottom",
```

```
                "value": "top-and-bottom"
              },
              {
                "label": "Top",
                "value": "top"
              },
              {
                "label": "Bottom",
                "value": "bottom"
              },
              {
                "label": "Center",
                "value": "center"
              }
            ]
          }
        }
      },
      {
        "label": "Top",
        "slug": "top",
        "types": ["gfx-global-variable", "css-length", "css-percentage"],
        "defaults": {
          "css-length": "0px",
          "css-percentage": "0%"
        },
        "defaultType": "css-length",
        "dependencies": [
          {
            "accessor": "./anchor",
            "values": ["top", "top-and-bottom"]
          }
        ]
      },
      {
        "label": "Translate-y",
        "slug": "translate_y",
        "types": ["gfx-global-variable", "css-length", "css-percentage"],
        "defaults": {
          "css-length": "0px",
          "css-percentage": "0%"
        },
        "defaultType": "css-length",
        "dependencies": [
          {
            "accessor": "./anchor",
            "values": ["center"]
          }
        ]
      },
      {
```

```
          "label": "Bottom",
          "slug": "bottom",
          "types": ["gfx-global-variable", "css-length", "css-percentage"],
          "defaults": {
            "css-length": "0px",
            "css-percentage": "0%"
          },
          "defaultType": "css-length",
          "dependencies": [
            {
              "accessor": "./anchor",
              "values": ["bottom", "top-and-bottom"]
            }
          ]
        },
        {
          "label": "Height",
          "slug": "height",
          "types": ["gfx-global-variable", "css-length", "css-percentage",
"form-select"],
          "defaults": {
            "css-length": "0px",
            "css-percentage": "0%"
          },
          "defaultType": "css-length",
          "dependencies": [
            {
              "accessor": "./anchor",
              "values": ["top", "bottom", "center"]
            }
          ],
          "parameterTypeBasedOptions": {
            "form-select": {
              "options": ["auto", "max-content", "min-content", "fit-content",
"stretch"]
            }
          }
        }
      ]
    },
    {
      "slug": "horizontal",
      "parametersPresentation": "row",
      "parameters": [
        {
          "label": "Horizontal Anchor",
          "slug": "anchor",
          "types": ["gfx-global-variable", "form-select"],
          "defaults": {
            "form-select": "left"
          },
```

```
      "defaultType": "form-select",
      "parameterTypeBasedOptions": {
        "form-select": {
          "options": [
            {
              "label": "Left & Right",
              "value": "left-and-right"
            },
            {
              "label": "Left",
              "value": "left"
            },
            {
              "label": "Right",
              "value": "right"
            },
            {
              "label": "Center",
              "value": "center"
            }
          ]
        }
      }
    },
    {
      "label": "Left",
      "slug": "left",
      "types": ["gfx-global-variable", "css-length", "css-percentage"],
      "defaults": {
        "css-length": "0px",
        "css-percentage": "0%"
      },
      "defaultType": "css-length",
      "dependencies": [
        {
          "accessor": "./anchor",
          "values": ["left", "left-and-right"]
        }
      ]
    },
    {
      "label": "Right",
      "slug": "right",
      "types": ["gfx-global-variable", "css-length", "css-percentage"],
      "defaults": {
        "css-length": "0px",
        "css-percentage": "0%"
      },
      "defaultType": "css-length",
      "dependencies": [
        {
```

```
              "accessor": "./anchor",
              "values": ["right", "left-and-right"]
            }
          ]
        },
        {
          "label": "Translate-x",
          "slug": "translate_x",
          "types": ["gfx-global-variable", "css-length", "css-percentage"],
          "defaults": {
            "css-length": "0px",
            "css-percentage": "0%"
          },
          "defaultType": "css-length",
          "dependencies": [
            {
              "accessor": "./anchor",
              "values": ["center"]
            }
          ]
        },
        {
          "label": "Width",
          "slug": "width",
          "types": ["gfx-global-variable", "css-length", "css-percentage",
"form-select"],
          "defaults": {
            "css-length": "0px",
            "css-percentage": "0%"
          },
          "defaultType": "css-length",
          "dependencies": [
            {
              "accessor": "./anchor",
              "values": ["left", "right", "center"]
            }
          ],
          "parameterTypeBasedOptions": {
            "form-select": {
              "options": ["auto", "max-content", "min-content", "fit-content",
"stretch"]
            }
          }
        }
      ]
    }
  ]
}
```

- SIZE_CONSTRAINTS

```
{
  "label": "Size Constraints",
  "slug": "size_constraint",
  "parametersPresentation": "row",
  "parameters": [
    {
      "label": "Min-Width",
      "slug": "min_width",
      "types": ["gfx-global-variable", "css-length", "css-percentage"],
      "defaults": {
        "css-length": "0px",
        "css-percentage": "0%"
      },
      "defaultType": "css-length"
    },
    {
      "label": "Max-Width",
      "slug": "max_width",
      "types": ["gfx-global-variable", "css-length", "css-percentage"],
      "defaults": {
        "css-length": "0px",
        "css-percentage": "100%"
      },
      "defaultType": "css-percentage"
    },
    {
      "label": "Min-Height",
      "slug": "min_height",
      "types": ["gfx-global-variable", "css-length", "css-percentage"],
      "defaults": {
        "css-length": "0px",
        "css-percentage": "0%"
      },
      "defaultType": "css-length"
    },
    {
      "label": "Max-Height",
      "slug": "max_height",
      "types": ["gfx-global-variable", "css-length", "css-percentage"],
      "defaults": {
        "css-length": "0px",
        "css-percentage": "100%"
      },
      "defaultType": "css-percentage"
    }
  ]
}
```

- PADDING

```
{
  "label": "Paddding",
  "slug": "padding",
  "parametersPresentation": "row",
  "parameters": [
    {
      "label": "Top",
      "slug": "top",
      "types": ["gfx-global-variable", "css-length", "css-percentage"],
      "defaults": {
        "css-length": "0px",
        "css-percentage": "0%"
      },
      "defaultType": "css-length"
    },
    {
      "label": "Right",
      "slug": "right",
      "types": ["gfx-global-variable", "css-length", "css-percentage"],
      "defaults": {
        "css-length": "0px",
        "css-percentage": "0%"
      },
      "defaultType": "css-length"
    },
    {
      "label": "Bottom",
      "slug": "bottom",
      "types": ["gfx-global-variable", "css-length", "css-percentage"],
      "defaults": {
        "css-length": "0px",
        "css-percentage": "0%"
      },
      "defaultType": "css-length"
    },
    {
      "label": "Left",
      "slug": "left",
      "types": ["gfx-global-variable", "css-length", "css-percentage"],
      "defaults": {
        "css-length": "0px",
        "css-percentage": "0%"
      },
      "defaultType": "css-length"
    }
  ]
}
```

- MARGIN

```
{
  "label": "Margin",
  "slug": "margin",
  "parametersPresentation": "row",
  "parameters": [
    {
      "label": "Top",
      "slug": "top",
      "types": ["gfx-global-variable", "css-length", "css-percentage"],
      "defaults": {
        "css-length": "0px",
        "css-percentage": "0%"
      },
      "defaultType": "css-length"
    },
    {
      "label": "Right",
      "slug": "right",
      "types": ["gfx-global-variable", "css-length", "css-percentage"],
      "defaults": {
        "css-length": "0px",
        "css-percentage": "0%"
      },
      "defaultType": "css-length"
    },
    {
      "label": "Bottom",
      "slug": "bottom",
      "types": ["gfx-global-variable", "css-length", "css-percentage"],
      "defaults": {
        "css-length": "0px",
        "css-percentage": "0%"
      },
      "defaultType": "css-length"
    },
    {
      "label": "Left",
      "slug": "left",
      "types": ["gfx-global-variable", "css-length", "css-percentage"],
      "defaults": {
        "css-length": "0px",
        "css-percentage": "0%"
      },
      "defaultType": "css-length"
    }
  ]
}
```

- TEXT_STYLING

```
{
  "label": "Text Styling",
  "slug": "text_style",
  "subSectionsPresentation": "column",
  "subsections": [
    {
      "slug": "font",
      "parametersPresentation": "row",
      "parameters": [
        {
          "slug": "family",
          "label": "Font Family",
          "types": ["gfx-global-variable", "css-font-family"],
          "defaults": {
            "css-font-family": "system-ui"
          }
        },
        {
          "slug": "size",
          "label": "Font Size",
          "types": ["gfx-global-variable", "css-length", "css-percentage"],
          "defaults": {
            "css-length": "14px",
            "css-percentage": "0%"
          },
          "defaultType": "css-length"
        },
        {
          "slug": "weight",
          "label": "Font Weight",
          "types": ["gfx-global-variable", "form-select", "form-number"],
          "defaults": {
            "form-select": "400",
            "form-number": "400"
          },
          "defaultType": "form-select",
          "parameterTypeBasedOptions": {
            "form-select": {
              "options": [
                {
                  "label": "Thin",
                  "value": "100"
                },
                {
                  "label": "Extra Light",
                  "value": "200"
                },
                {
                  "label": "Light",
                  "value": "300"
```

```
                },
                {
                  "label": "Normal",
                  "value": "400"
                },
                {
                  "label": "Medium",
                  "value": "500"
                },
                {
                  "label": "Semi Bold",
                  "value": "600"
                },
                {
                  "label": "Bold",
                  "value": "700"
                },
                {
                  "label": "Extra Bold",
                  "value": "800"
                },
                {
                  "label": "Black",
                  "value": "900"
                }
              ]
            }
          }
        }
      ]
    },
    {
      "slug": "format",
      "parametersPresentation": "row",
      "parameters": [
        {
          "slug": "is_italicized",
          "label": "Italicize",
          "types": ["gfx-global-variable", "form-checkbox"],
          "defaults": {
            "form-checkbox": "true"
          }
        },
        {
          "slug": "is_underlined",
          "label": "Underline",
          "types": ["gfx-global-variable", "form-checkbox"],
          "defaults": {
            "form-checkbox": "true"
          }
        },
```

```
        {
          "slug": "color",
          "label": "color",
          "types": ["gfx-global-variable", "css-color"],
          "defaults": {
            "form-checkbox": "#000000FF"
          }
        }
      ]
    }
  ]
}
```

- BORDER_STYLING

```
{
  "slug": "border",
  "label": "Border Styling",
  "parametersPresentation": "row",
  "parameters": [
    {
      "label": "Border Thickness",
      "slug": "thickness",
      "types": ["gfx-global-variable", "css-length"],
      "defaults": {
        "css-length": "0px"
      }
    },
    {
      "label": "Border Radius",
      "slug": "radius",
      "types": ["gfx-global-variable", "css-length"],
      "defaults": {
        "css-length": "0px"
      }
    },
    {
      "label": "Border Color",
      "slug": "color",
      "types": ["gfx-global-variable", "css-color"],
      "defaults": {
        "css-color": "#000000FF"
      }
    }
  ]
}
```

- SHADOW_STYLING

```
{
  "slug": "shadow",
  "label": "Shadow Styling",
  "subSectionsPresentation": "column",
  "subsections": [
    {
      "slug": "",
      "parametersPresentation": "row",
      "parameters": [
        {
          "slug": "is_visible",
          "label": "Show Shadow",
          "types": ["gfx-global-variable", "form-checkbox"],
          "defaults": {
            "form-checkbox": "false"
          }
        },
        {
          "slug": "color",
          "label": "Shadow Color",
          "types": ["gfx-global-variable", "css-color"],
          "defaults": {
            "css-color": "#000000FF"
          }
        }
      ]
    },
    {
      "slug": "",
      "parametersPresentation": "row",
      "parameters": [
        {
          "slug": "x_offset",
          "label": "X Offset",
          "types": ["gfx-global-variable", "css-length"],
          "defaults": {
            "css-length": "0px"
          }
        },
        {
          "slug": "y_offset",
          "label": "Y Offset",
          "types": ["gfx-global-variable", "css-length"],
          "defaults": {
            "css-length": "0px"
          }
        },
        {
          "slug": "blur_radius",
          "label": "Blur Radius",
```

```
          "types": ["gfx-global-variable", "css-length"],
          "defaults": {
            "css-length": "0px"
          }
        }
      ]
    }
  ]
}
```

- BACKGROUND

```
{
  "label": "Background",
  "slug": "background",
  "parameters": [
    {
      "label": "Fill",
      "slug": "fill",
      "types": ["gfx-global-variable", "css-color", "css-image"],
      "defaults": {
        "css-color": "#000000FF",
        "css-image": "url(\"\")"
      },
      "defaultType": "css-color"
    }
  ],
  "parametersPresentation": "row",
  "subSectionsPresentation": "column",
  "subsections": [
    {
      "slug": "",
      "parametersPresentation": "row",
      "parameters": [
        {
          "label": "Background Size",
          "slug": "size",
          "types": ["gfx-global-variable", "form-select", "css-length",
"css-percentage"],
          "parameterTypeBasedOptions": {
            "form-select": {
              "options": ["cover", "contain", "auto"]
            }
          },
          "defaults": {
            "form-select": "contain"
          },
          "defaultType": "form-select"
        },
```

```
        {
          "label": "Background Repeat",
          "slug": "repeat",
          "types": ["gfx-global-variable", "form-select"],
          "parameterTypeBasedOptions": {
            "form-select": {
              "options": ["repeat", "no-repeat", "space", "round", "repeat-x",
"repeat-y"]
            }
          },
          "defaults": {
            "form-select": "repeat"
          },
          "defaultType": "form-select"
        }
      ]
    }
  ]
}
```

If a user wanted to, they can create a custom macro with the macroName of CUSTOM10 and they can copy the value of BOUNDS and paste for the value of CUSTOM10 inside the sectionMacros property.

GraphicDesignerParameter object

Typescript definition

```
type GraphicDesignerParameter = {
  label?: string // optional
  slug: string
  description?: string // optional
  types: GraphicDesignerParameterType[]
  defaultType?: GraphicDesignerParameterType // optional
  defaults?: {
    [type in GraphicDesignerParameterType]?: string // optional
  } // optional
  parameterTypeBasedOptions?: {
    'css-length'?: CssLengthParamOptions // optional
    'css-percentage'?: CssPercentageOptions // optional
    'css-angle'?: CssAngleOptions // optional
    'css-color'?: CssColorOptions // optional
    'css-image'?: CssImageOptions // optional
    'form-text'?: FormTextOptions // optional
    'form-number'?: FormNumberOptions // optional
    'form-select'?: FormSelectOptions // optional
```

```
    'form-checkbox'?: FormCheckboxOptions // optional
    'form-toggle'?: FormToggleOptions // optional
  } // optional
  dependencies?: {
    accessor: string
    values: string[] // or
  }[] // optional // and
}
```

JSON example

```
{
  "label": "Fill",
  "slug": "fill",
  "types": ["css-color", "css-image"],
  "defaults": {
    "css-color": "#000000FF",
    "css-image": "url(\"\")"
  },
  "defaultType": "css-color"
}
```

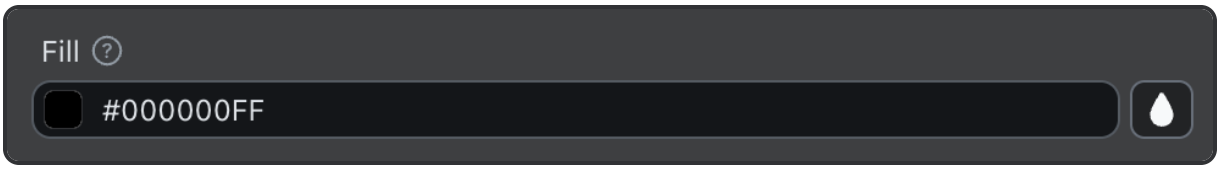


Table reference

Property	Type	Required
label	string	optional
slug	string	
description	string	optional
types	Array of GraphicDesignerParameterType	
defaultType	GraphicDesignerParameterType	optional
defaults	*Refer to code snippet*	optional
parameterTypeBasedOptions	*Refer to code snippet*	optional
dependencies	*Refer to code snippet*	optional

GraphicDesignerFormSection object

Typescript definition

```
type GraphicDesignerFormSection = {
  label?: string // optional
  slug: string
  description?: string // optional
  parametersPresentation?: 'flex-grid' | 'column' | 'row' // optional
  parameters?: (GraphicDesignerParameter | ParameterMacroReference)[] // optional
  subSectionsPresentation?: 'accordion' | 'tabs' | 'column' | 'row' // optional
  subsections?: (GraphicDesignerFormSection | SectionMacroReference)[]
// optional
}
```

JSON example

```
{
  "label": "Test container 1",
  "slug": "tc_1",
  "description": "This is a test GraphicDesignerFormSection",
  "parametersPresentation": "column",
  "parameters": [
    {
      "label": "Fill",
      "slug": "fill",
      "types": ["css-color", "css-image"],
      "defaults": {
        "css-color": "#000000FF",
        "css-image": "url(\"\")"
      },
      "defaultType": "css-color"
    }
  ],
  "subSectionsPresentation": "tabs",
  "subsections": ["PADDING", "MARGIN"]
}
```

Table Reference

Property	Type	Required
label	string	optional
slug	string	
description	string	optional
parametersPresentation	"flex-grid"	"column" | "row" | optional
parameters	Array of GraphicDesignerParameter and ParameterMacroReference	optional
subSectionsPresentation	"accordion"	"tabs" | "column" | "row" | optional
subsections	Array of GraphicDesignerFormSection and SectionMacroReference	optional

SectionMacroReference type

This is a reference to either a default **section** macro preset or a custom section macro created by the user. The user can either provide the macro name as a simple string or supply an object to overwrite both the label and slug.

Typescript definition

```
type SectionMacroReference =
  | string
  | {
      macroName: string
      label?: string
      slug?: string
    }
```

Table reference

Property	Type	Required
macroName	string	"BOUNDS" | "SIZE_CONSTRAINT" | "PADDING" | "MARGIN" | "TEXT_STYLING" | "BORDER_STYLING" | "SHADOW_STYLING" | "BACKGROUND"
label	string	optional
slug	string	optional

ParameterMacroReference object

This is a reference to a custom **parameter** macro created by the user. The user can either provide the macro name as a simple string or supply an object to overwrite both the label and slug.

Typescript definition

```
type ParameterMacroReference =
  | string
  | {
      macroName: string
      label?: string
      slug: string // optional
    }
```

Table reference

Property	Type	Required
macroName	string	
label	string	optional

Property	Type	Required
slug	string	optional

Reference

GraphicDesignerParameterType options

Table reference

Option name	Type
css-length	string
css-percentage	string
css-angle	string
css-color	string
css-image	string
css-font-family	string
form-text	string
form-number	string
form-select	string
form-checkbox	string
form-toggle	string
gfx-global-variable	string

CSS Parameter options

CssLengthParamOptions

Typescript definition

```
type CssLengthParamOptions = {
  units: ('px' | 'em' | 'vh' | 'vw' | 'cqw' | 'cqh' | 'cqi')[]
// empty means all
  constraints?: {
    [unit in 'px' | 'em' | 'vh' | 'vw' | 'cqw' | 'cqh' | 'cqi']: {
      min?: number // optional
      max?: number // optional
      step?: number
// optional; sets the increment for the unit's stepper control
    }
  }[] // optional
}
```

Table reference

Property	Type	Required
units	Array of "px" | "em" | "vh" | "vw" | "cqw" | "cqh" | "cqi"	
constraints	*Refer to code snippet*	optional

CssPercentageOptions

Typescript definition

```
type CssPercentageOptions = {
  constraints?: {
    min?: number
    max?: number
  }
}
```

Table reference

Property	Type	Required
constraints	*Refer to code snippet*	optional

CssAngleOptions

No support yet

CssColorOptions

Typescript definition

```
undefined
```

CssImageOptions

Typescript definition

```
undefined
```

CssFontFamily

Typescript definition

```
undefined
```

Form parameter options

FormTextOptions

Typescript definition

```
undefined
```

FormNumberOptions

Typescript definition

```
undefined
```

FormSelectOptions

Typescript definition

```
type FormSelectOptions = {
  options:
    | {
        value: string
        label: string
      }[]
    | string[]
```

```
}
```

JSON example

```
{
  "options": [
    {
      "value": "500",
      "label": "Medium"
    }
  ]
}
```

Table reference

Property	Type	Required
options	Array of string or Array of `{"value": string, "label": string}`	

FormCheckboxOptions

Typescript definition

```
undefined
```

FormToggleOptions

Typescript definition

```
undefined
```

Special Gfx parameter options

GfxGlobalVariableOptions

Typescript definition

```
undefined
```

Combiner Graphic Type

Overview

The Combiner graphic type is a composite graphic. Rather than consuming a category of meeting data like the other graphic types, it layers multiple existing graphics - and static images - into a single output, letting you control each layer's position, scale, and stacking order.

This lets you deliver one combined graphic URL to a downstream video mixer instead of importing and arranging several separate graphic layers there.

Creating a Combiner

Combiner graphics are created from their own entry point. On the Graphics Dashboard, click the **New Combiner** button rather than the **New Graphic** button used for the other graphic types.

The Create New Combiner dialog presents a gallery of Combiner templates. A template defines a starting arrangement of *component slots*, where each slot is either a graphic slot or an image slot, pre-positioned and scaled on the 1920×1080 canvas (shown in the layout preview). You can filter the gallery by All Templates, Zoom Templates, or My Templates.

Select a template, give your Combiner a name (which determines its output URL), optionally add a description, and create it. You can turn an existing Combiner into a reusable template at any time using the **Save As Template** action in its row menu on the Graphics Dashboard.

The Combiner Editor

Opening a Combiner shows a Graphics Editor with a single tab labeled **Combiner**, in place of the four tabs used by the other graphic types.

The preview window behaves like any other graphic's preview, with one addition: the **Edit Outline** control (Auto / On / Off) draws an outline around each component so you can see its bounds while arranging. *Auto* shows the outlines while you are editing, *On* always shows them, and *Off* hides them.

Adding Components: Graphics and Images

A Combiner displays component layers of two kinds:

- **Graphic components** - any of your other graphics (Combiner graphics themselves are excluded). Add them with the **Add Graphic** button.
- **Image components** - static images from your Asset Library. Add them with the **Add Image** button.

You can display up to five graphic components and up to five image components on a single Combiner. The buttons show the current counts, for example "Add Graphic (2/5)".

If your Combiner was created from a template, it may start with unassigned placeholder slots. An unassigned slot shows a dashed outline and a **Select Graphic** or **Select Image** button; use it to fill the slot with one of your graphics or an image.

The Component Settings Box

Each component has a settings box. Collapsed, it shows the component's type icon and name on the left, and the following controls on the right:

- The **eye** icon toggles the component's visibility on this Combiner. **This only affects how it displays on this Combiner, and does not change the graphic's state anywhere else in the app.**
- The **chevron** expands or collapses the box to reveal the position controls.
- The **...** menu provides **Replace with graphic**, **Replace with image**, and **Remove**.

Position and Scale

Expanding a component reveals its **Width**, **Height**, **X Position**, **Y Position**, and **Scale**. Adjust them with the steppers or by typing a value, then press Enter or click away to apply.

These properties apply only to the component within this Combiner, and do not change the underlying graphic anywhere else in the app.

Layering (Z-Index)

Component boxes are stacked in z-index order. Drag and drop a settings box to reorder it: the top box renders on top (the highest z-index) and the bottom box renders beneath the others. The new order is applied to the live output when you drop the box.

Simulating Component Data

A component graphic with no live data - for example, a chat scroll before any messages arrive - appears empty. To preview a Combiner with representative data:

- Open a single component's simulation tools with the **Simulation Settings** button in its expanded box. The **Show simulated data on live output** option is locked on here, because simulated data would otherwise not be visible on the Combiner.
- Use **Manage All Simulations**, below the component list, to control every component at once: set them all to live or simulated data, and start or stop all of their simulators together.

Opening a Component's Editor and Exporting

The **Open Editor** button in an expanded component opens that graphic's own full Graphics Editor.

From the Graphics Dashboard, a Combiner's **...** row menu also lets you **Export Combiner** to a `.zgfx` file and **Save As Template**.

Counter Graphic Type

Overview

The Counter graphic type displays live meeting engagement statistics as a set of compact "counters." Each counter shows a single number — or the elapsed meeting time — with an optional icon and label. Examples include total chat messages, questions asked, participants, emoji reaction tallies, poll responses, hands raised, and video-on or muted counts.

A single Counter graphic can show one counter or many, arranged together as cards.

Creating a Counter

Create a Counter the same way as most graphic types: click **New Graphic** on the Graphics Dashboard and choose the **Counter** type. Select a preset to start from — **Unstyled Base** (no visual theme), **Basic Dark**, or **Basic Light** — then name your graphic and create it.

Choosing What to Display — the Graphic Settings Tab

Open the Counter in the Graphics Editor and select the **Graphic Settings** tab. This is where you choose which counters appear and how often they refresh.

- **Counters list** — check the counters you want to show. Each row shows the counter's icon, label, and a description of its data source. Use **Select All** / **Unselect All** to toggle them together.
- **Reordering** — checking or unchecking a counter automatically moves it to the boundary between the checked and unchecked counters, so enabled counters

stay clustered together. Drag a counter by its handle to fine-tune the order within that group. The order is saved to this graphic, and new graphics start in alphabetical order.

- **Editing a counter** — the edit button opens an icon picker with **Defaults**, **Zoom Icons**, and **Emojis** sections to choose the counter's icon, plus controls for icon color and label text. **Restore Default** returns the icon, color, and label to their defaults in one action.
- **Polling Interval** — how often the participant-derived counters refresh (1, 2, 5, 10, or 15 seconds). A help popover lists recommended values by meeting size: 1 second for under 1,000 participants, 2 seconds for 1,001–5,000, and longer intervals for larger webinars.
- **Elapsed Time** — this counter is a manual stopwatch with **Play/Pause** and **Reset** controls. The timer does not start automatically. Its value is saved to the graphic, so it keeps counting across meetings — including while the meeting isn't active — and stays in sync in real time for everyone viewing that graphic. Each Counter graphic has its own independent timer. It automatically stops at a 4-hour cap.

Note that participant-based counters (participants, attendees, video and mute counts, and similar) may not display accurately in webinars with more than 5,000 participants.

Styling — the Graphic Designer Tab

The **Graphic Designer** tab controls the appearance of the counters, including:

- **Layout Direction** — whether multiple counter cards are arranged in a row or a column.
- **Counter Content Order and Direction** — the order of each counter's icon, value, and label, and whether they stack vertically or sit in a row.
- **Vertical Alignment**, **Horizontal Alignment**, and **Vertical Offset** — applied per element (icon, value, and label) within each counter card.

- **Element Spacing** and **Edge Spacing** — the gap between the icon, value, and label, and the padding around them within the card.
- **Counter Card / Counter Icon / Counter Value / Counter Label styling** — background, size, colors, and fonts, plus a Universal Icon Color option.

As with the other graphic types, advanced users can edit the underlying HTML and CSS in the **Template Editor** tab and adjust the form itself in the **Designer Config** tab. See the Counter graphic template input for the values exposed to the template.

Participant List Graphic Type

Overview

The Participant List graphic type displays the meeting or webinar roster as a scrolling list of cards, one per participant. It can show everyone, or a filtered/grouped view — for example only raised hands, only muted participants, or a breakdown of Yes / No voting responses.

The graphic automatically adapts its role terminology to the session: in a meeting it labels people as Hosts, Co-Hosts, and Participants; in a webinar it labels them as Hosts, Co-Hosts, Panelists, and Attendees, matching Zoom's own terminology.

Creating a Participant List

Create a Participant List the same way as most graphic types: click **New Graphic** on the Graphics Dashboard and choose the **Participant List** type. Select a preset to start from — **Unstyled Base** (no visual theme), **Basic Dark**, or **Basic Light** — then name your graphic and create it.

Choosing What to Display — the Graphic Settings Tab

Open the Participant List in the Graphics Editor and select the **Graphic Settings** tab.

- **List Title** — optional text shown above the list in its own card.
- **Participant List Type** — a dropdown of 12 list types, each with a description of what it shows:
  - **Everyone (no filter)** — all meeting participants.
  - **Hand Raised** — only participants with a raised hand.
  - **Active Emoji Reactions** — participants currently showing a reaction, grouped by reaction; reactions clear after about 10 seconds, matching Zoom.
  - **Grouped by Role** — everyone, grouped under role headers (Hosts/Co-Hosts/Participants, or Hosts/Co-Hosts/Panelists/Attendees in a webinar).
  - **Waiting Room** — participants currently in the waiting room.
  - **Grouped by Video On/Off**, **Video On**, **Video Off** — filtered or grouped by camera state.
  - **Grouped by Muted/Unmuted**, **Muted**, **Unmuted** — filtered or grouped by audio state.
  - **Yes / No Voting** — groups everyone into Yes, No, and Not Yet Voted based on Zoom's non-verbal feedback. Votes persist until a participant clears or changes them.
- **Sort Participants** — **Order met** (raised-hand order, reaction order, or join order depending on the list type) or **Alphabetical**.
- **Exclude Webinar Attendees** — in a webinar, hides attendees so the list shows only hosts, co-hosts, and panelists. Not available for the Waiting Room list type.
- **Polling Interval** — how often participant data refreshes (1, 2, 5, 10, or 15 seconds). A help popover lists recommended values by meeting size: 1 second under 1,000 participants, 2 seconds for 1,001–5,000, 5 seconds for 5,001–25,000, 10 seconds for 25,001–50,000, and 15 seconds beyond that. This interval is shared with the Counter graphic's participant-derived counters.

Note that participant data may not display accurately in webinars with more than 5,000 participants.

Pagination and Scrolling

When the list has more rows than fit in the graphic's bounds, it splits into pages ("waves") and automatically scrolls from one to the next, with an upward scroll animation similar to the Chat Scroll graphic. If a group spans more than one wave, its group header repeats at the top of the continuation page. The time between waves is set in the Graphic Designer tab's **Time Between Scroll Waves** control (1–60 seconds).

Styling — the Graphic Designer Tab

The **Graphic Designer** tab controls the appearance of the list, including:

- **List Customization** — Name Style (Full Name / Initials / First Initial + Last Name), Profile Image (None / Initials Avatar / Real Photo with initials fallback), Row Numbering (hidden or shown), and Time Between Scroll Waves.
- **Row, Avatar, Name, and Row Number styling** — background, size, colors, and fonts for each part of a participant row.
- **Group Header styling** — text color, background, sizing, font weight, text transform, and card padding/border radius for the stylized group header cards.
- **List Title Card styling** — background, text color, fonts, and card padding/border radius for the list title.
- **Status Icon styling** — size and color of the per-row status icon (used for emoji reactions and Yes/No voting indicators).

As with the other graphic types, advanced users can edit the underlying HTML and CSS in the **Template Editor** tab and adjust the form itself in the **Designer Config** tab. See the Participant list graphic template input for the values exposed to the templates.