

chat-window

<https://playdocs1.orangeriver-ad055946.westus2.azurecontainerapps.io/play-docs/docs/ui-recipes/chat-window>

Chat Window

The component is a comprehensive chat interface that provides message display, real-time input, custom icons, and automatic scrolling. It features a flexible message system, keyboard shortcuts, and responsive design for building interactive chat applications, customer support systems, and messaging interfaces.

```
<ava-chat-window>
```

How to use

```
import { AavaChatWindowComponent, ChatMessage, ChatWindowIcon, } from "@aava/play-core";
```

```
import { AavaChatWindowComponent, ChatMessage, ChatWindowIcon, } from "@aava/play-core";
```

Basic Usage

A chat window with message history, input area, and custom icons for sending messages and attachments.

```
<aava-chat-window [messages]="messages()" [placeholder]="placeholder" [disabled]="disabled" [rows]="rows"
[icons]="chatIcons" (messageSent)="onMessageSent($event)" (iconClicked)="onIconClick($event)"></aava-chat-
window>
```

```
<aava-chat-window [messages]="messages()" [placeholder]="placeholder" [disabled]="disabled" [rows]="rows"
[icons]="chatIcons" (messageSent)="onMessageSent($event)" (iconClicked)="onIconClick($event)"></aava-chat-
window>
```

```
@ViewChild(AavaChatWindowComponent) chatWindow!: AavaChatWindowComponent; messages = signal<ChatMessage[]>
([]); placeholder = "Type your message here..."; disabled = false; rows = 3; chatIcons: ChatWindowIcon[] = [
  { name: 'paperclip', slot: 'icon-start', size: 16, color: '#2563eb', click: () =>
    this.onFileAttach() }, { name: 'wand-sparkles', slot: 'icon-end', size: 16, color:
    '#2563eb', click: () => this.onMagicAction() }, { name: 'send', slot: 'icon-end', size:
    16, color: '#2563eb', click: () => this.onSendClick() } ]; onMessageSent(messageText: string) {
  // Add user message const userMessage: ChatMessage = { id: Date.now().toString(), text: messageText,
  timestamp: new Date().toLocaleString('en-US', { hour: 'numeric', minute: '2-digit', hour12:
  true }) + ' | Today', isUser: true }; this.messages.update(messages => [...messages,
  userMessage]); // Add bot response after delay setTimeout(() => { const botMessage: ChatMessage = {
  id: (Date.now() + 1).toString(), text: this.generateBotResponse(messageText), timestamp: new
  Date().toLocaleString('en-US', { hour: 'numeric', minute: '2-digit', hour12: true
  }) + ' | Today', isUser: false, avatar: ' ' }; this.messages.update(messages =>
  [...messages, botMessage]); }, 1000); } onIconClick(event: { icon: ChatWindowIcon, currentMessage: string
}) { console.log('Icon clicked:', event.icon.name, 'Current message:', event.currentMessage); if
(event.icon.name === 'send') { // User can add custom logic here before sending console.log('Send icon
clicked, message:', event.currentMessage); // The send will be handled by the icon's click handler or
manually trigger } }
```

```
@ViewChild(AavaChatWindowComponent) chatWindow!: AavaChatWindowComponent; messages = signal<ChatMessage[]>
([]); placeholder = "Type your message here..."; disabled = false; rows = 3; chatIcons: ChatWindowIcon[] = [
  { name: 'paperclip', slot: 'icon-start', size: 16, color: '#2563eb', click: () =>
    this.onFileAttach() }, { name: 'wand-sparkles', slot: 'icon-end', size: 16, color:
    '#2563eb', click: () => this.onMagicAction() }, { name: 'send', slot: 'icon-end', size:
    16, color: '#2563eb', click: () => this.onSendClick() } ]; onMessageSent(messageText: string) {
  // Add user message const userMessage: ChatMessage = { id: Date.now().toString(), text: messageText,
  timestamp: new Date().toLocaleString('en-US', { hour: 'numeric', minute: '2-digit', hour12:
  true }) + ' | Today', isUser: true }; this.messages.update(messages => [...messages,
  userMessage]); // Add bot response after delay setTimeout(() => { const botMessage: ChatMessage = {
  id: (Date.now() + 1).toString(), text: this.generateBotResponse(messageText), timestamp: new
  Date().toLocaleString('en-US', { hour: 'numeric', minute: '2-digit', hour12: true
  }) + ' | Today', isUser: false, avatar: ' ' }; this.messages.update(messages =>
  [...messages, botMessage]); }, 1000); } onIconClick(event: { icon: ChatWindowIcon, currentMessage: string
}) { console.log('Icon clicked:', event.icon.name, 'Current message:', event.currentMessage); if
(event.icon.name === 'send') { // User can add custom logic here before sending console.log('Send icon
clicked, message:', event.currentMessage); // The send will be handled by the icon's click handler or
manually trigger } }
```

Features

Message Display

- Message History: Scrollable message container with user and bot messages
- Message Styling: Different styling for user and bot messages
- Timestamps: Automatic timestamp display for each message
- Auto-scroll: Automatic scrolling to the latest message
- Custom Avatars: Support for user avatars in messages

Input System

- Real-time Input: Live text input with placeholder support
- Keyboard Shortcuts: Enter to send, Shift+Enter for new line
- Custom Icons: Configurable icons for different actions
- Icon Slots: Start and end icon slots for flexible positioning
- Disabled State: Support for disabled input state

User Experience

- Responsive Design: Adapts to different screen sizes
- Smooth Scrolling: Smooth auto-scroll behavior
- Custom Scrollbars: Styled scrollbars for better UX
- Focus Management: Proper focus handling for accessibility
- Loading States: Support for loading and processing states

Configuration

- Flexible Messages: Support for different message types and content
- Icon Configuration: Customizable icons with click handlers
- Input Configuration: Configurable placeholder, rows, and styling
- Event Handling: Comprehensive event system for all interactions

API Reference

Inputs

Property	Type	Default	Description
messages	ChatMessage[]	[]	Array of chat messages to display
placeholder	string	'Type a message'	Placeholder text for the input field
disabled	boolean	false	Whether the chat window is disabled
icons	ChatWindowIcon[]	[]	Array of icons to display in the input area
rows	number	3	Number of rows for the textarea input
messages			
ChatMessage[]			
[]			
placeholder			
string			

'Type a message'
disabled
boolean
false
icons
ChatWindowIcon[]
[]
rows
number
3

Outputs

Property	Type	Description
messageSent	EventEmitter	Emitted when a message is sent
iconClicked	EventEmitter<{icon: ChatWindowIcon, currentMessage: string}>	Emitted when an icon is clicked
messageSent		
EventEmitter<string>		
iconClicked		
EventEmitter<{icon: ChatWindowIcon, currentMessage: string}>		

Methods

Method	Description
triggerSend()	Public method to trigger message sending
getCurrentMessage()	Public method to get the current message text
triggerSend()	
getCurrentMessage()	

Interfaces

interface ChatMessage { id: string; // Unique message identifier text: string; // Message content timestamp: string; // Message timestamp isUser: boolean; // Whether message is from user or bot avatar?: string; // Optional avatar URL }
interface ChatMessage { id: string; // Unique message identifier text: string; // Message content timestamp: string; // Message timestamp isUser: boolean; // Whether message is from user or bot avatar?: string; // Optional avatar URL }
interface ChatWindowIcon { name: string; // Icon name click?: () => void; // Optional click handler size?:

```
number; // Icon size (default: 16)  color?: string; // Icon color (default: '#2563eb')  slot: "icon-start" | "icon-end"; // Icon position}
```

```
interface ChatWindowIcon {  name: string; // Icon name  click?: () => void; // Optional click handler  size?: number; // Icon size (default: 16)  color?: string; // Icon color (default: '#2563eb')  slot: "icon-start" | "icon-end"; // Icon position}
```

CSS Classes

The component provides several CSS classes for styling:

Class Name	Description
.chat-window	Main chat window container
.chat-main	Main chat content wrapper
.messages-container	Scrollable messages area
.message-wrapper	Individual message wrapper
.user-message	User message styling
.bot-message	Bot message styling
.message-content	Message content container
.message-card	Message card styling
.message-text	Message text styling
.message-timestamp	Message timestamp styling
.input-area	Input area container

.chat-window

.chat-main

.messages-container

.message-wrapper

.user-message

.bot-message

.message-content

.message-card

.message-text

.message-timestamp

.input-area

CSS Custom Properties

The component uses CSS custom properties for theming:

Property	Description
--surface-secondary	Secondary surface color for scrollbar
--border-color	Border color for input and scrollbar
--surface-background	Background color for input area
--text-tertiary	Tertiary text color for scrollbar
--text-secondary	Secondary text color for scrollbar
--border-radius-full	Full border radius for input

--surface-secondary

--border-color

```
--surface-background
```

```
--text-tertiary
```

```
--text-secondary
```

```
--border-radius-full
```

Best Practices

Message Management

- Unique IDs: Ensure each message has a unique identifier
- Timestamp Format: Use consistent timestamp formatting
- Message Length: Consider message length limits for better UX
- Loading States: Show typing indicators for better user experience

User Experience

- Auto-scroll: Always scroll to bottom for new messages
- Keyboard Support: Provide keyboard shortcuts for common actions
- Visual Feedback: Clear visual distinction between user and bot messages
- Responsive Design: Ensure chat works well on mobile devices

Performance

- Message Limits: Consider limiting displayed messages for performance
- Efficient Rendering: Use `trackBy` functions for large message lists
- Memory Management: Clean up old messages to prevent memory leaks
- Debounced Input: Consider debouncing input for better performance

Accessibility

- Screen Reader Support: Ensure messages are properly announced
- Keyboard Navigation: Full keyboard support for all interactions
- Focus Management: Proper focus handling for input and icons
- ARIA Labels: Provide appropriate ARIA labels for interactive elements

Accessibility Guidelines

Semantic Structure

The component provides proper semantic structure:

- Message Roles: Proper roles for user and bot messages
- Input Labels: Clear labels for input fields
- Icon Descriptions: Proper descriptions for interactive icons
- Focus Management: Logical focus order through chat elements

Screen Reader Support

- Message Announcements: Clear announcements of new messages
- Input Feedback: Proper feedback for input actions

- Icon Descriptions: Descriptive labels for all icons
- Status Updates: Clear status updates for loading and sending

Keyboard Navigation

- Tab Order: Logical tab order through chat elements
- Enter Key: Enter key for sending messages
- Icon Activation: Keyboard activation for all icons
- Focus Indicators: Clear focus indicators for keyboard users

Color and Contrast

- WCAG Compliance: All text and interactive elements meet WCAG AA contrast ratios
- High Contrast Mode: Component works with system high contrast settings
- Color Independence: Information is not conveyed by color alone
- Visual Hierarchy: Clear visual distinction between message types

Responsive Behavior

Mobile Adaptations

The chat window automatically adapts to mobile screens:

- Touch Optimization: Optimized touch targets for mobile interaction
- Keyboard Handling: Proper mobile keyboard behavior
- Viewport Adaptation: Adapts to different mobile viewport sizes
- Scroll Behavior: Smooth scrolling on mobile devices

Breakpoint Behavior

- Desktop (>768px): Full chat interface with all features
- Mobile (≤768px): Compact layout with optimized spacing
- Input Sizing: Responsive input area sizing
- Icon Sizing: Appropriate icon sizes for different screens

Content Considerations

- Message Length: Messages adapt to different screen widths
- Input Height: Input area adjusts based on content
- Scroll Performance: Optimized scrolling for mobile devices
- Touch Targets: Adequate touch target sizes for mobile