ChatSession Control Reference

Display a chat conversation thread between an end-user and an AI model.

Reference **Controls** **Interaction** ChatSession

The **TChatSession** control (shown in the Designer Components Panel under the *Interaction* category as "ChatSession") is the FrameworX surface for an AI chat panel inside a Display. It belongs to the *Portable* rendering target — the same control renders identically in the WPF Rich Client and in the HTML5/WASM Web Client.

In FrameworX 10.1.5 the control is **zero-substrate** for the standard pattern: drop a TChatSession on a Display, drop a TextBox bound to a Query tag, drop a Button with the `ChatRequest` Action, and the live conversation appears multi-turn and role-bubbled without any DataTable, UserType, or Script wiring on the adopter side.

Overview
- Minimum operator chat panel — the zero-substrate pattern

Visual contract
- Layout — single-column-left
- Bubble shape — selectable TTextBlock
- Color palette — theme tokens, not RGB
- Sizing — auto-size to wrapped content
- Corner shaping — BubbleCornerRadius
- Auto-scroll — AutoScrollToBottom
- Pending-indicator UX
- Error states — bubbles in the chat track

Role discrimination — tolerant matching

One chat session per client connection
- Targeting a specific chat session

Cross-target — WPF and HTML5

Lifecycle actions

Properties
- Inherited properties not authored on this control

Advanced — bring-your-own transcript

See also

Overview

TChatSession renders an auto-scrolling vertical thread of per-role message bubbles. The control pulls the conversation directly from the runtime's transcript cache (per client connection and chat control) — the same cache the ChatRequest Action Reference action populates on every chat turn. There is no transcript Tag to bind, no DataTable to configure, no UserType to define, and no Server Script glue to write.

This is the standard pattern for every chat surface in FrameworX: operator chat, knowledge-graph assistant, SupportAI debug, partner portal, training tutor. Each surface composes the same five elements (TChatSession + Input TextBox + Send/Clear/Compress Buttons + scalar Query/Reply Tags) and re-skins independently.

Minimum operator chat panel — the zero-substrate pattern

Three elements, two scalar tags, one Action. No scripting:

1. A **TextBox** whose write binding is a String tag (e.g. `@Tag.Chat/Query`).
2. A **Button** with an Action dynamic of type `ChatRequest`. Set the Action's **Query** field to the Query tag and the **Return** field to a JSON tag (e.g. `@Tag.Chat/Reply`).
3. A **TChatSession** control on the same Display panel. Leave `ItemsSourceLink` empty — the control auto-resolves to the live transcript for the current client.

When the operator clicks the Button, the live conversation appears in the TChatSession: the user's question lands immediately as an accent-tinted bubble (no waiting for the LLM), a rotating ? pending indicator with an elapsed-seconds counter follows it, and the assistant reply bubble replaces the indicator when the model finishes. Follow-up questions on the same panel retain context — the model sees the full transcript and can refer back to earlier turns.

For a complete working solution showing this pattern end-to-end — six live plant tags as grounding context, three quick-prompt buttons, the AI Engine tile configured for a local Ollama model — see Chat Session Example.

Visual contract

The control's render surface is opinionated — the visual shape is pinned by the platform so every chat surface in FrameworX reads consistently. Authors do not style the bubbles individually; they style the control's outer chrome (background, border) and the platform paints the bubbles per role.

Layout — single-column-left

Every bubble — user, assistant, and unknown-role — left-edges in a single column. **Color discriminates role; alignment does not.** The layout follows the conversational-AI cluster convention (Claude Code, Claude.ai, ChatGPT, Cursor, GitHub Copilot, Linear AI) rather than the peer-to-peer messenger convention (iMessage, WhatsApp, Slack DMs). Each bubble caps at ~75 % of the control's rendered width; the rest of the row is empty so the eye scans one ragged-right margin instead of two.

Bubble shape — selectable TTextBlock

Bubble bodies render as `T.Wpf.RunControls.TTextBlock` elements (the FrameworX-native control that inherits `System.Windows.Controls.TextBlock`). Chrome is suppressed by the control — no border, no focus rectangle, no caret, no keyboard tab-stop.

Bubble text is **selectable**. Operators drag-select bubble text with the mouse, then press **Ctrl-C** to copy to the clipboard, or right-click and pick **Copy** from the context menu. Selection works symmetrically on user and assistant bubbles. The capability is enabled via `TextBlock.IsTextSelectionEnabled` on WPF .NET 4.8; on HTML5/WASM (OpenSilver) it degrades gracefully — the bubbles render correctly and selection activates as soon as OpenSilver surfaces the underlying property.

Color palette — theme tokens, not RGB

Role	Background	Foreground (text)	Border
User	*AccentBrush* (platform accent surface)	*ThemeWhiteBrush* (high-contrast text on accent)	*DefaultBorder*
Assistant	*ShadeBrush* (neutral shade)	*TextForeground*	*DefaultBorder*
Unknown (Role matches neither `UserRoleValue` nor `AIRoleValue`)	*ShadeBrush*	*TextForeground*	*DefaultBorder*

The control resolves these theme tokens dynamically — a live theme switch repaints every bubble without a transcript reload, via the platform's ThemeDP attached-property surface. Authors should not hardcode bubble RGB values; the theme tokens are the contract.

Sizing — auto-size to wrapped content

Bubbles auto-size to their wrapped content (`Height = NaN`, `MinHeight = 22`) so multi-line replies grow vertically without being clipped. The horizontal cap is ~75 % of `TChatSession.ActualWidth` (the outer control), so wide screens render correctly without re-skinning and narrow screens wrap rather than horizontally scroll.

Corner shaping — `BubbleCornerRadius`

Default 14 pixels. Set to `0` for square bubbles, higher values for a more conversational appearance. Applies uniformly to user and assistant bubbles.

Auto-scroll — `AutoScrollToBottom`

Default `true`. On every transcript update (new turn lands, ChatCompress replaces, initial open with non-empty cache) the inner ScrollViewer scrolls to the bottom so the freshest message is in view. Set to `false` when the user is reviewing historical transcript and the scroll position should be preserved.

Pending-indicator UX

When the operator clicks the Send Button (ChatRequest), TChatSession appends two visuals immediately — before the LLM round-trip begins:

- An **immediate user-echo bubble** carrying the operator's typed query, styled with the user-role palette. The operator sees their message land at once instead of staring at an empty panel for 3–10 seconds.
- A **pending row** with a rotating **?** glyph (one revolution per ~1.2 seconds) and an elapsed-seconds counter that ticks `0s, 1s, 2s, …`.

Both visuals are replaced by the authoritative server-emitted bubble pair when the LLM reply lands — no flicker between the echo and the server-snapshot bubble.

Error states — bubbles in the chat track

When the LLM is disabled, unreachable, returns a non-OK envelope, or times out, the platform writes BOTH the user's question AND a paired **error-role** assistant message into the per-(clientGuid, chatName) transcript cache. The populator renders both as bubbles. The user's question persists; the error text appears as a danger-tinted bubble immediately below it. No separate error band, no toast, no disappearing question — the chat thread is the surface for everything.

Error bubbles use the platform's *AlarmHighPriority* background token (the canonical alarm-tier red, load-bearing on every shipped theme) with *ThemeWhiteBrush* foreground for high contrast and *DefaultBorder* border. Layout, padding, margin, corner radius, max-width, and selection support are identical to user and assistant bubbles — only the palette differs.

Gate state	What the user sees
LLM endpoint unreachable (HTTP error)	User-question bubble + error bubble: "LLM endpoint HTTP error: <detail>".
Master kill-switch off (`SolutionCapabilities[LocalAI].Enabled=false`)	User-question bubble + error bubble: "Local AI master kill-switch (SolutionCapabilities[LocalAI].Enabled) is off."
Auto-warm failure (Ollama not installed or not running)	User-question bubble + error bubble with actionable install/start guidance.

ModelOptions `EnableRuntimeMCP` bit (0x02) off	User-question bubble + error bubble naming the disabled bit.
Request wall-clock timeout (60 s budget)	User-question bubble + error bubble: "Chat request timed out."

> **JSON envelope still available for audit.** The Reply JSON tag continues to carry the full envelope (`status`, `text`, `toolTrace`, `latencyMs`, `warnings[]`) for adopters who want it for logging, audit, or diagnostic display. The chat surface does NOT depend on JsonString-extracting any envelope field — the platform writes the bubble text directly to the transcript cache, robust across LLM providers because the envelope is FrameworX-emitted, not LLM-emitted.

Transcript and conversation memory. The transcript cache keeps every turn for each connected client, successful replies and error pairs alike, regardless of the `ModelOptionsChatHistory` bit. The panel always shows the live conversation. The bit controls only whether earlier turns are sent back to the model as conversation memory. With the bit clear, the model receives the current turn only, and when the chat has earlier turns the reply carries a warning that conversation memory is off.

Role discrimination — tolerant matching

The control declares two role-marker properties:

- `UserRoleValue` — default `user`. Identifies user-role bubbles.
- `AIRoleValue` — default `assistant`. Identifies assistant-role bubbles.

Matching is **case-insensitive** (`OrdinalIgnoreCase`) and also tolerates common alias synonyms on each side. A bubble whose `Role` column matches the declared property OR appears in the role's alias set renders with that role's palette; bubbles whose Role matches neither fall through to the unknown-role branch (neutral palette).

Side	Default property value	Alias set (case-insensitive)
User	`user`	`user`, `operator`, `human`, `question`, `you`, `me`, `customer`, `client`, `person`, `prompt`
Assistant	`assistant`	`assistant`, `ai`, `bot`, `agent`, `response`, `answer`, `plant ai`, `system ai`, `system`, `model`, `copilot`, `reply`

The defaults plus the alias whitelist mean most adopters never need to set `UserRoleValue` or `AIRoleValue` explicitly — the runtime writes `user` and `assistant` on every row, and both defaults match. Override the properties only when feeding the control from a custom transcript whose role labels do not match the defaults or any alias.

For persona labels visible on the Display (e.g. "Operator", "Plant AI"), add a separate TextBlock above the transcript — do not push the persona text into `UserRoleValue` / `AIRoleValue`. The control's contract is to match what the runtime writes; the persona is a Display-level affordance.

One chat session per client connection

The conversation transcript is held per client connection, not per control. Every `TChatSession` on every Display of one connected client shows the same conversation, and the lifecycle actions (`ChatRequest`, `ChatClear`, `ChatCompress`) all act on that single transcript.

Two consequences worth designing around:

- Placing a second `TChatSession` on a Display does not create a second conversation when the two controls share the same Uid — which is the case for every control dropped from the toolbox — and both render the same transcript. If the controls carry different Uids (for example after a copy/paste, which assigns a new Uid, or a Uid renamed by hand), chat actions target only the first control and the others show an empty transcript. Do not give chat controls on the same Display different Uids.
- `ChatClear` has no per-panel granularity. A diagnostic popup that clears on close also wipes a main chat panel open elsewhere for the same client.

> The `Object` field on `ChatClear`, `ChatCompress` and `ChatCopyToTag` is not used and is ignored. In releases before 10.1.6, a non-empty value was sent to the server as a conversation key and addressed a conversation no panel displayed, so the action appeared to do nothing. From 10.1.6 the field has no effect.

Targeting a specific chat session

Independent chat sessions on one Display — each control holding its own transcript, with the action able to name the session it targets — are not supported in this release. The behaviour above applies: chat controls on one Display share one conversation per client connection.

Cross-target — WPF and HTML5

TChatSession is shared source — the same control compiles into the WPF Rich Client (.NET 4.8) and into the HTML5 Web Client (OpenSilver / WASM). Author wiring does NOT differ between the two targets. Visual parity holds for the full layout, the color palette, the auto-scroll, the auto-size height, and the pending-indicator. Selection + Ctrl-C copy is active today on the WPF target; on the HTML5 target it degrades gracefully — no error, just no selection — until the underlying OpenSilver TextBlock surfaces the relevant property.

Lifecycle actions

TChatSession is a rendering control. The three Display Actions below operate on the transcript that TChatSession displays. Attach them to Buttons on the same Display panel for a complete operator chat surface.

Action	What it does
ChatRequest Action Reference	Sends the operator's query to the LLM, maintains the conversation transcript across turns, dispatches platform tools the model may request, refreshes the TChatSession surface on reply landing. The primary action for an operator chat panel.
ChatClear Action Reference	Wipes the transcript for the addressed chat session immediately, without calling the LLM. Use for "Start new conversation" buttons and shift-handover resets. Synchronous and idempotent.
ChatCompress Action Reference	Summarises the current transcript via one LLM call and atomically replaces it with the summary. Preserves semantic continuity while reducing token cost on long conversations. Asynchronous; takes 3–10 seconds on a local model.

Properties

Property set auto-extracted from `ControlSchemas.json`.

Property	Type	Default	Description
`Left`	`Double`	0	X position from left edge *(unit: pixels)*
`Top`	`Double`	0	Y position from top edge *(unit: pixels)*
`Width`	`Double`	100	Element width *(unit: pixels)*. Cleared to auto-size on Loaded.
`Height`	`Double`	100	Element height *(unit: pixels)*. Cleared to auto-size on Loaded.
`AutoScrollToBottom`	`Boolean`	`true`	When `true`, the inner viewport scrolls to the newest bubble on every transcript update. Recommended for live chat. Set `false` when reviewing history.
`UserRoleValue`	`String`	`user`	String the control treats as the user-role marker. Case-insensitive + alias whitelist (see "Role discrimination" above).
`AIRoleValue`	`String`	`assistant`	String the control treats as the assistant-role marker. Case-insensitive + alias whitelist (see "Role discrimination" above).
`BubbleCornerRadius`	`Double`	14	Corner rounding of each message bubble, in pixels. Applies uniformly to user and assistant bubbles.
`ItemsSourceLink`	`String`	*(empty)*	**Advanced.** When empty, the control auto-resolves to the live server-side transcript for the current client (recommended). When set to a custom DataTable Tag reference (e.g. `@Tag.MyDataTable`), the control reads from that Tag instead — the bring-your-own-transcript escape hatch (see below).
`ItemsSourceOption`	`Enum`	`DataTable`	Pinned to `DataTable` by the control's ctor. Authors do not change this.
`Background`	`Color`	*(themed)*	Background color of the control's outer chrome (#AARRGGBB). Themed by default — OMIT to use theme.
`Foreground`	`Color`	*(themed)*	Default text/foreground color of the outer chrome (#AARRGGBB). Themed by default. Note: bubble foreground is driven by the per-role palette and is NOT controlled by this property.
`BorderBrush`	`Color`	*(themed)*	Outer border color (#AARRGGBB).
`BorderThickness`	`String`	*(themed)*	Outer border thickness (single value or `left,top,right,bottom`).
`MaxItemsLink`	`String`	`100` at fresh-drop	Caps the maximum number of bubbles the control renders. Useful for very long transcripts where the visible history must be bounded. Fresh-drop default is `100`; existing Displays preserve whatever the .dbsln carries.

Inherited properties not authored on this control

The control inherits a Designer-canvas / preview-mode sample-rendering surface from its parent class (`DesignQuantity`, `PreviewQuantity`, `DesignElements`, `PreviewElements`, `PreviewMargin`, `DesignSingleType`, `PreviewSingleType`, `PreviewSettings`, `DesignWidth`, `PreviewWidth`, `DesignHeight`, `PreviewHeight`, `DesignMargin`, `DesignSettings`, `ReloadItemsLink`, `ContainerPanel`). These properties are not authored on TChatSession — they describe a generic list-of-elements panel surface, not a chat session whose container shape and data source are pinned by the control itself. The Designer property panel suppresses them by default.

Advanced — bring-your-own transcript

The zero-substrate path is the recommended pattern for ~95 % of chat surfaces. The *bring-your-own-transcript* escape hatch exists for the remaining cases — a frozen archive Display, a filtered conversation view, a cross-solution merge.

Set `ItemsSourceLink` to a DataTable Tag reference (e.g. `@Tag.MyDataTable`). The control bypasses the zero-substrate auto-resolve and reads from that Tag instead. The DataTable schema must contain at minimum:

- An `Element` column carrying the fully-qualified element type name (e.g. `T.Wpf.RunControls.TTextBlock`) — identifies the per-row bubble shape.
- A `LinkedValue` column carrying the message text (each cell is the bubble's content).
- A `Role` column carrying the role string — `user`, `assistant`, or any value in the alias whitelist.

When the DataTable updates (rows added, rows removed, contents changed), bind `ReloadItemsLink` to a pulse tag so the control re-reads on demand.

Adopters using a custom DataTable Tag continue to work unchanged against the zero-substrate ship — the escape hatch is purely additive.

See also

- Chat Session Example — the canonical minimal solution demonstrating the zero-substrate pattern end-to-end.
- ChatRequest Action Reference — the primary Action paired with this control.
- ChatClear Action Reference — transcript wipe action.
- ChatCompress Action Reference — transcript summarization action.
- Local AI — parent reference covering the full Local AI feature surface.
- Local AI Configuration Reference — endpoint configuration, ModelOptions bits, tool-category gates.
- Local AI Reply Envelope Schema — the JSON shape that lands on the Reply tag.

In this section...
