

Permalink URI:

vscode://joelabenhaim.aicode-ja/chat?sessionId=d39e59a5fe2349a68e30f5f6d10c2794-VS-V%3%A9rifier-SESSION-<USER>

[Verify]SPECIF FINALISER LIVESESSION POUR LES SESSIONS PROMPT NORMALES 20260712 204731</USER>

<ASSISTANT>

I'm thinking...

</ASSISTANT>

<ASSISTANT>

🔧  running code interpreter: [details](#)

🔧  scanning recent files

🔧  scanning git: 📁 "État Git courant", 📁 "Fichiers modifiés", 📁 "Volume des changements"

🔧  reading project: 📁 g "Docs live session", 📁 g "Docs Prompt ciblées"

🔧  scanning git: 📁 "Commit courant", 📁 "Historique récent", 📁 "Périmètre depuis base", 📁 "Statistiques depuis base"

🔧  reading project: 📁 f "Doc isomorphisme", 📁 f "Doc thinking Prompt", 📁 f "Doc Queue B", 📁 f "Doc persistance fallback", 📁 f "Doc EasyAgents", 📄 [liveSession.programming-doc-readme.md](#), 📄 [easyAgents.programming-doc-readme.md](#), 📄 [isomorphism-invariant-rule-between-streaming-and-rehydration-of-chat-sessions.programming-doc-readme.md](#), 📄 [thinking-bubble-and-hourglass-streaming-rules-invariants.programming-doc-readme.md](#), 📄 [prompt-stream-dispatch-lock.programming-doc-readme.md](#), 📄 [fallback-persist-and-title.programming-doc-readme.md](#), 📄 e "Ancien abort source", 📄 e "Ancien état windows", 📄 e "Ancienne clé stream", 📄 e "Identité ouverture replay", 📄 e "Identité tentative stream", 📄 e "Hydratation locale", 📄 e "Erreur body Slimbk", 📄 e "Contexte origine stream", 📄 e "Stop source activité",

📄 d "usePromptLiveSession",

📄 d "useAiStream",

📄 d "LiveSessionRegistryService",

📄 d "NormalPromptLiveSessionSourceService",

📄 d "PromptLiveSessionPanelCoordinator",

📄 d "SidebarQueryActivityRuntimeService",

📄 d "parsePromptLiveSessionPanelQuery", 📄 d "dispatchPromptLiveSessionBootstrap", 📄 h "Registre

liveSession", 📄 h "Source Prompt normale", 📄 h "Coordination panneau", 📄 h "Démarrage

transactionnel", 📄 h "Replay Prompt normal", 📄 e "Tentative ouverture", 📄 e "Génération

ouverture",

📄 [LiveSessionRun.type.ts](#),

📄 [LiveSessionPanel.type.ts](#),

📄 [LiveSessionOpenResult.type.ts](#),

📄 [SidebarStore.ts](#),

📄 [AiStreamStartArgs.type.ts](#),

📄 [PromptLiveSession.types.ts](#),

📄 [NormalPromptLiveSession.adapter.ts](#),

📄 [PromptLiveSessionBootstrap.adapter.ts](#),

📄 [usePromptLiveSession.hook.ts](#),

📄 [applyHydratedPromptSessionState.util.ts](#),

📄 [parseSlimbkStreamBodyError.util.ts](#), 📄 [LiveSessionRecord.type.ts](#), 📄 [LiveSessionRegistry.service.ts](#) [L1-

L380], 📄 [LiveSessionRegistry.service.ts](#) [L620-

L1110],

📄 [LiveSessionService.ts](#),

📄 [LiveSessionOpen.service.ts](#),

📄 [LiveSessionSourceLifecycle.service.ts](#),

📄 [NormalPromptLiveSessionSource.service.ts](#), 📄 [LiveSessionRegistry.service.ts](#) [L350-

L650], 📄 [LiveSessionRegistry.service.ts](#) [L1090-L1136], 📄 e "Préparation Open", 📄 e "Revalidation

Open", 📄 [aiStopSuffix.util.ts](#), 📄 e "Synthèse erreur source", 📄 e "Erreur système source", 📄 e "Interruption

source", 📄 [NormalPromptLiveSessionSource.service.test.ts](#) [L280-

L480],

📄 [extTypedMessageBus.util.ts](#),

[extMessageBus.util.ts](#),
[iaStreamHandlers.ts](#),
[liveSessionStreamHandlers.ts](#),
[stream.endpoint.ts](#),
[abortSource.endpoint.ts](#), [abort.endpoint.ts](#), [open.endpoint.ts](#), [stream.endpoint.ts](#), e "Échec arrêt son source", e "Erreur auxiliaire source",
[PromptQueryDoneSound.manager.ts](#),
[PromptLiveSessionPanelQuery.util.ts](#),
[PromptLiveSessionPanelCoordinator.service.ts](#),
[PromptLiveSessionDetachedStop.service.ts](#),
[SidebarQueryRunningBridge.ts](#),
[SidebarStore.queryWindow.listeners.manager.ts](#),
[SidebarStore.queryWindow.mutations.manager.ts](#),
[SidebarStore.rehydrate.manager.ts](#),
[PromptThreadCleanup.manager.ts](#), [delete.endpoint.ts](#), [useAiStream.hook.ts](#) [L1-L340], [useAiStream.hook.ts](#) [L520-L1035], [useAiStreamQueue.hook.ts](#), [usePromptStreamDispatchGateQueue.hook.ts](#) [L1-L340], [usePromptStreamDispatchGateQueue.hook.ts](#) [L840-L1010], e "Identité active Queue B", e "Comparaison tentative Queue B", [usePromptStreamDispatchGateQueue.hook.ts](#) [L340-L840], e "Queue A tentative stale", e "Queue B successeur", [PromptStreamDispatchGateQueue.test.ts](#) [L1020-L1142], [useAiStream.hook.ts](#) [L300-L520], e "Bootstrap Prompt live", e "Barrière fallback Prompt", e "Finalisation tentative Prompt", e "Query live explicite",
[usePromptRunCommands.hook.ts](#),
[preparePromptStreamingTurn.util.ts](#),
[preparePromptContinuationStreaming.util.ts](#),
[usePromptCancelController.hook.ts](#),
[usePromptRunController.finalization.hook.ts](#), [usePromptThinkingLifecycle.hook.ts](#), [Prompt.tsx](#) [L1-L340], [Prompt.tsx](#) [L520-L760], [Prompt.tsx](#) [L900-L1120], [PromptInput.tsx](#), [SidebarActivityQuery.tsx](#), e "Armement barrière fallback"
  scanning git: ? "Usages barrière fallback"
  reading project: [Prompt.tsx](#) [L340-L520], [Prompt.tsx](#) [L1160-L1260], [Prompt.tsx](#) [L1260-L1443],
[openPanel.util.ts](#),
[OpenedPanelsRegistry.util.ts](#),
[closePromptPanel.util.ts](#), [SidebarStore.handlers.manager.ts](#), [SidebarUiMessages.ts](#)
  scanning git: ? "Diff Queue A", ? "Diff Queue B", ? "Diff fichiers générés", ? "Contrôle whitespace"
  reading project: [SlimbkResult.type.ts](#), e "Décodage V1 chaîne", [aiStopSuffix.util.test.ts](#) [L120-L280], [callLocalApi.util.ts](#), h "Client stream erreurs", [callLocalApi.generated.ts](#) [L1-L230], [aiStopSuffix.util.test.ts](#) [L80-L130], e "Stop puis échec replay", e "Retry après Stop", [usePromptLiveSession.normalPrompt.test.tsx](#) [L1-L313], e "Refus Stop transport", [useAiStream.hook.promptSourceIdentity.test.ts](#) [L217-L316], e "Erreur RPC Stop",
[usePromptRunController.hook.ts](#),
[EasyAgentsBatchRunRetry.service.ts](#),
[EasyAgentsBatchLiveSessionAdapter.service.ts](#),
[EasyAgentsBatchRunChildExecution.service.ts](#), [EasyAgentsBatchRunManager.service.ts](#)
  scanning git: ? "Ajouts any", ? "Ajouts eslint disable", ? "Ajouts TODO", ✗ ? "Imports dynamiques prod", ✗ ? "Require production", ? "Casts unknown ajoutés", ? "Imports dynamiques prod", ? "Require production", ? "Détails casts commandes"
  reading project: [SlimbkEndpoint.ts](#), [Slimbk.ts](#), e "Envoi header stream", e "Erreur post header", [MessageRegistry.ts](#) [L1-L180], r "openCentralPanel", [OrchMessagesMemoryAssistantPatch.ts](#) [L1-L260], [OrchMessagesMemoryAssistantPatch.ts](#) [L260-L420], e "Flag inscription source", [EasyAgentsBatchRunExecution.service.ts](#) [L280-L410], e "Claim pendant

[cleanup](#)",  [PromptLiveSessionPanelCoordinator.service.test.ts](#) [L320-L460],  e ["Source concurrente retry"](#),  [EasyAgentsBatchRunRetry.guards.service.test.ts](#) [L1-L220],  [usePromptSessionLocator.hook.ts](#),  e ["SourceId fourni au start"](#),  [VscButton.tsx](#) [L1-L240],  [SidebarStore.core.manager.ts](#) [L1-L240]

Vérification du commit courant

Joel, la spécification est d'un niveau d'architecture rarement aussi précis — elle est presque assez détaillée pour demander elle-même une revue de code et un café au scheduler ☕.

La vérification porte sur le **HEAD exact 2002ecd666012f3e211c53a88f3b21d71dbeaec4**, intitulé `feat(live-session): keep prompt runs alive across panel close and reopen`. L'arbre Git est propre.

J'ai notamment relu les documents de maintenance du sous-système :

- [Architecture liveSession](#)
- [Architecture EasyAgents](#)
- [Invariants d'isomorphisme Prompt](#)
- [Invariants thinking](#)
- [Architecture Queue B](#)
- [Barrière fallback persist](#)

1) La spécification entière est-elle implémentée ?

Réponse

 **Non, pas strictement à 100 %.**

L'implémentation contient bien **tous les grands sous-systèmes fonctionnels** demandés :

- identité exacte `locator + turnId + sourceId` ;
- réservation avant les opérations asynchrones ;
- replay backlog + tail ;
- modes `replaceTurnAssistants` et `continueAssistant` ;
- claims destructifs et génération destructive ;
- contexte de webview d'origine ;
- continuation d'une génération normale après fermeture ;
- projection Activité `open/closed` ;
- Stop détaché ;
- reconnexion Prompt avec composer local ;
- handoff direct ;
- `preserveLocal` ;
- terminal durable ;
- adaptation EasyAgents ;
- suppression et Retry sous claim ;
- documentation et couverture de tests très étendues.

Je n'ai trouvé **aucun grand bloc architectural totalement absent**.

En revanche, plusieurs garanties strictes de la spécification ne sont pas complètement fermées, notamment :

1. la récupération après certains échecs de Stop ;
2. la revalidation d'un claim destructif pendant un nettoyage asynchrone ;
3. la course entre le précheck EasyAgents Retry et l'acquisition du claim ;
4. la garde d'identité avant toute mutation interne de Queue B ;
5. certains cas transport après terminal V1 ;
6. l'agrégation finale des erreurs du handler source.

Estimation

- **Couverture structurelle : environ 97 %**
- **Conformité comportementale stricte : environ 90 %**
- **Complétude globale estimée : environ 94 %**

Divergences non bloquantes

- Le concept demandé sous le nom `replayOpenAttemptId` existe sous la forme interne `openAttemptRef`. La sémantique est correcte ; seul le nom littéral diffère.
- Le commit modifie aussi des fichiers nécessaires qui ne figuraient pas dans le récapitulatif « exact » de la spécification, notamment :
 - [useAiStreamQueue.hook.ts](#)
 - [usePromptStreamDispatchGateQueue.hook.ts](#)

• [PromptFallbackPersistBarrier.testHarness.tsx](#)

Ces ajouts sont cohérents avec les invariants demandés. Il s'agit surtout d'une **inexactitude de l'inventaire de la spécification**, pas d'une faute fonctionnelle.

2) L'implémentation actuelle fait-elle ce qu'elle doit faire selon la spécification ? Pour chaque fonctionnalité technique demandée, est-elle complète ?

Bloc technique demandé	État	Complétude estimée	Preuves vérifiées
Identité exacte des runs	✓ Complet	100 %	sourceId est obligatoire, inclus dans la run key mais pas dans la runId. Les anciens symboles streamKey et canAbortSource n'ont pas été supprimés. L'union EasyAgents/normalPrompt, les politiques de fermeture de ligne requise/interdite et l'inventaire privé sont centralisés dans les Branches live, terminal, not_ready, not_found, pruned, et opaque dans LiveSessionOpenResult.type.ts .
Descripteur partagé strict	✓ Complet	100 %	Réservation synchrone, promotion sans second bump, refus de fermeture destructive et événements immuables dans LiveSessionOpenResult.type.ts .
Contrat Open strict et enveloppe policy-aware	✓ Complet	99 %	Époque de run, génération destructive, claim et active pointer dans LiveSessionOpen.service.ts .
Réservations, époques, retraits, tombstones et claims	✓ Complet	98 %	replaceTurnAssistants filtre uniquement les assistants dans LiveSessionOpen.service.ts .
Open avec double revalidation	✓ Complet	98 %	baseline immuable dans LiveSessionOpen.service.ts .
Replay modes et baselines	✓ Complet	98 %	Priming avant enregistrement, settlement unique, fermeture déterministe dans LiveSessionSourceLifecycle.service.ts .
Lifecycle lazy Readable partagé	✓ Complet	98 %	La persistance meta.error → append → forward, les erreurs sont gérées dans NormalPromptLiveSessionSource.service.ts .
Source Prompt normale et persistance des erreurs	⚠ Presque complet	93 %	imparfait.
Protocole premier-header V1	⚠ Presque complet	92 %	Encodeur/décodeur stricts dans aiStopSuffix.util.ts .
Erreur Slimbk post-header dans le body	✓ Complet	98 %	terminal initial est correct, mais une erreur postérieure à ce point est possible. Détection avant la logique AiStreamedChunk dans parseStreamedChunk.ts les deux handlers extension.
Origine webview exacte non sérialisée	✓ Complet	100 %	StreamHandlerOriginContext contient le HtmlPage dans extTypedMessageBus.util.ts ; il est construit par StreamHandlerOriginContext .
FIFO extension et finalisation post-forwarding	⚠ Presque complet	94 %	L'ordre append/Activity/son/emit/finalisation est visible dans le coordinateur n'est cependant pas intégrée à l'agrégation d'activités.
Coordination fermeture/réouverture du panneau	⚠ Partiel	86 %	Priorité explicite, injection normalPrompt, FIFO par locator dans PromptLiveSessionPanelCoordinator.service.ts chaque await destructif.
Parse strict des queries live	✓ Complet	99 %	Doublons, inconnus, partiels, locator ambigu, lease fraîche dans PromptLiveSessionPanelQuery.ts .
Activité session-oriented	✓ Complet	98 %	L'union Idle/open, Running/observer/open et Running/souvent les transitions et timers exacts sont portés par SidebarQuery.ts .
Réhydratation Activité	✓ Complet	97 %	Fusion panneaux, réservations, records et runtimes dans SidebarQuery.ts fermés et Idle fermés sont exclus.
Stop détaché dans Activité	✓ Complet	97 %	Rendu uniquement pour source/closed avec SquareStop, key sourceId dans SidebarActivityQuery.tsx ; claim dans PromptLiveSessionDetachedStop.service.ts .
Démarrage frontend transactionnel	✓ Complet	97 %	start() refuse sans mutation, génère les identités après la mutation dans useAiStream.hook.ts . Les commandes committe dans usePromptRunCommands.hook.ts .
Identités Queue A/Queue B	⚠ Partiel	90 %	Les entrées portent bien sourceId + streamInstance tard pour un item Queue A arrivé après un reset.

Bloc technique demandé	État	Complétude estimée	Preuves vérifiées
Open attempt, Retry et handoff direct	✓ Presque complet	95 %	Une identité Open indépendante, un seul Open en vol, timers sont présents dans usePromptLiveSession.hook.ts .
<code>preserveLocal</code> sans lecture du brouillon	✓ Complet	100 %	Le mode ne lit que meta et messages, sans accéder à prompt dans applyHydratedPromptSessionState.util.ts .
Composer normalPrompt et Stop unique	✓ Complet	98 %	Composer visible, édition après armement, Send verrouillé dans adaptateur normalPrompt , Prompt.tsx et PromptI .
Barrière fallback persist	✓ Complet	98 %	Barrière {turnId, streamInstanceId}, clear sur ack dans Prompt.tsx .
Continuation et thinking	✓ Complet	98 %	Plan pur, assistant réutilisé, thinking UID frais et séparation dans preparePromptContinuationStreaming.util .
Son exact par tentative	✓ Complet	98 %	État par runRef, soundAttemptId, dédup terminale et au dans PromptQueryDoneSound.manager.ts .
Adaptation EasyAgents	✓ Presque complet	94 %	sourceId=turnId, lifecycle partagé et politique observe enfant .
Retry EasyAgents destructif sous claim	⚠ Partiel	88 %	Le claim est bien tenu jusqu'au finally dans EasyAgen .
Delete durable sous claim	✓ Presque complet	94 %	est contrôlée trop tôt, avant plusieurs opérations asynchrones. Claim exact, fermeture, suppression disque, éviction après Delete . Sa sûreté complète dépend toutefois de la correction.
Documentation et tests ciblés	✓ Très étendu	96 %	La couverture est remarquable : registre, source normale, c isomorphisme et Retry. Les scénarios correspondant aux c

Conclusion de la question 2

✓ Le comportement principal demandé est présent et cohérent.

⚠ Il reste toutefois des courses de faible fréquence mais à fort impact qui empêchent de déclarer l'implémentation totalement conforme.

3) Y a-t-il des bugs ?

Réponse

✗ Oui. J'ai identifié quatre défauts certains et trois lacunes contractuelles conditionnelles.

ID	Sévérité	Certitude	Bug vérifié
B1	● Haute	Confirmé	Un Stop accepté avant l'ouverture du replay peut laisser Prompt définitivement bloqué. Dans usePromptLiveSession.hook.ts, sourceStopPending à true après un Stop accepté et n'est réinitialisé que lorsque le mode devient le RPC Stop est refusé/échoue. Si la source est retirée faute de baseline et Op ensuite not_found, le mode devient liveSessionReplayFailed, mais retryHandoff() refuse de fonctionner tant que sourceStopPending. L'adaptateur désactive en plus Retry via stopEnabled=false. Résultat : Stop désactivé, Send verrouillé, composer non interactif. Le nettoyage détaché ne revalide pas un nouveau claim destructif après les opérations asynchrones. PromptLiveSessionPanelCoordinator.service.ts vérifie le claim uniquement avant d'appeler PromptThreadCleanup.manager.ts. Le nettoyage vérifie correctement l'identité du mapping après chaque await, mais pas la g d'un claim acquis entre-temps. Le coordinateur poursuit ensuite cleanup son, retraits des marqueurs sans nouvelle revalidation. Une seconde suppression ou peut donc croiser ce cleanup. Le précheck EasyAgents Retry peut devenir périmé avant l'acquisition d'un claim. Dans EasyAgentsBatchRunRetry.service.ts, hasActiveSound et le test de query manuelle sont exécutés avant le guard Git et la préparation comportent plusieurs await. Une source Prompt manuelle peut démarrer pendant Le claim est ensuite acquis sans recheck, puis le run nouvellement démarré es
B2	● Haute	Confirmé	
B3	● Haute	Confirmé	

ID	Sévérité	Certitude	Bug vérifié
B4	● Moyenne	Confirmé	enfant supprimé. Un échec du RPC Stop consomme définitivement la capacité de Stop pen continue. useAiStream.hook.ts met <code>transportStopAvailable=false</code> au premier <code>await</code>, puis absorbe refus et erreurs sans restaurer cette capacité. usePromptCancelController.hook.ts garde alors <code>abortAckPending=true</code> tant que <code>isStreaming=true</code>. L'utilisateur retenter Stop avant la fin naturelle du stream. Un ancien item Queue A peut encore modifier l'état interne de Queue B a reset. Queue A conserve ses entrées ; <code>reset()</code> ne la vide pas. Si une entrée a dans <code>enqueueMergedForAttempt()</code> après le reset, Queue B lui attribue la et exécute <code>gateToolCallsChunk()</code> avant que la garde d'identité aval ne la ancien <code>tool_calls</code> de codeframe peut ainsi verrouiller <code>PromptStreamLoc</code> tentative suivante. Le test actuel couvre une continuation déjà engagée dans C une arrivée tardive depuis Queue A. Un premier <code>meta.error</code> envoyé par le header V1 peut masquer une er ultérieure. endpoint source Prompt émet immédiatement un header non-200. Slimbk termine alors le stream, et client généré Slimbk appelle <code>onError</code> puis source lève ensuite une erreur après ce terminal, cette erreur ne peut plus atteindre iaStreamHandlers.ts ; l'outcome peut donc rester <code>completed</code> de <code>transportError</code>. Les erreurs finales du coordinateur ne sont pas agrégées avec les erreurs transport/auxiliaires déjà collectées. Dans iaStreamHandlers.ts, <code>notifyPromptSourceSettle</code> du mécanisme d'accumulation. S'il échoue, il masque les erreurs préexistantes l'agrégation déterministe demandée.
B5	● Moyenne	Lacune contractuelle démontrable	
B6	● Moyenne	Lacune transport conditionnelle	
B7	● Faible à moyenne	Confirmé	

Tests de régression manquants correspondant aux bugs

Les suites sont très riches, mais elles ne couvrent pas encore précisément :

- Stop accepté avant baseline, puis retraite `retiredMissingBaseline`, puis Open `not_found` ;
- Stop RPC refusé ou rejeté pendant que `isStreaming` reste vrai ;
- acquisition d'un nouveau claim pendant un `await` du cleanup détaché ;
- source manuelle démarrante pendant le guard Git/préparation d'un `Retry EasyAgents` ;
- item Queue A ancien arrivant **après** le reset de Queue B ;
- premier terminal `meta.error` V1 suivi d'un `throw transport` ;
- erreur coordinateur combinée avec une erreur transport déjà mémorisée.

Conclusion de la question 3

Oui, des corrections sont nécessaires avant de considérer cette tranche comme finalisée pour la production.

4) Quelle est mon opinion sur le code source et la spécification ?

Opinion sur le code source

✅ Très bonne architecture générale.

Les qualités principales sont :

- séparation nette entre registre générique, politique produit, transport, panneau, Activité et frontend ;
- identité exacte présente dans toutes les couches importantes ;
- types stricts et unions discriminées solides ;
- claims opaques et générations indépendantes bien conçus ;
- très bonne discipline autour de la persistance avant `append` ;
- `preserveLocal` réellement opaque aux données du composer ;
- documentation technique alignée avec les concepts du code ;
- couverture de tests particulièrement importante ;
- commentaires techniques abondants et utiles ;
- conservation correcte du comportement `EasyAgents` existant.

Les faiblesses principales sont :

- le système possède plusieurs états d'autorité parallèles : registre, runtime Activité, SidebarStore, panneau, queue frontend, tentative Open et transport ;
- certaines garanties ne sont protégées que par l'ordre attendu des appels, pas par une garde mécanique au point de mutation ;
- les erreurs finales et les rollbacks de Stop ne disposent pas d'un contrat transactionnel aussi fort que le démarrage ;
- les méthodes de nettoyage ont encore besoin d'une revalidation uniforme après chaque frontière asynchrone ;
- plusieurs fichiers critiques dépassent mille lignes, notamment le registre et le hook de stream, ce qui augmente le coût de preuve des invariants.

Avis global sur le code : 9/10 pour l'architecture, environ 8/10 pour la fermeture complète des courses.

Opinion sur la spécification

✅ **La spécification est excellente et exceptionnellement rigoureuse.**

Elle réussit particulièrement bien sur :

- l'identification des autorités et identités ;
- les races ABA ;
- l'ordre des barrières ;
- la distinction terminal data / transport ;
- les effets destructifs ;
- l'isomorphisme stream/hydratation ;
- les politiques différentes normalPrompt/EasyAgents ;
- la liste détaillée des tests attendus.

Ses limites sont :

1. son inventaire de fichiers présenté comme « exact » n'inclut pas tous les fichiers effectivement nécessaires ;
2. la récupération après un RPC Stop refusé ou interrompu n'est pas explicitement définie ;
3. la revalidation d'un claim **acquis pendant** un cleanup déjà commencé méritait un invariant encore plus explicite ;
4. le protocole V1 ne décrit pas suffisamment comment conserver une erreur transport survenue après un premier terminal envoyé comme header non-200 ;
5. la tranche atomique est cohérente fonctionnellement, mais son ampleur rend la preuve d'absence de course extrêmement difficile.

Avis global sur la spécification : 9,5/10.

5) Résumer le rapport dans un tableau

TL-DR

Axe	Verdict	Estimation	Résumé
Couverture de la fonctionnalité	✅ Très élevée	94–96 %	Tous les grands blocs de la tranche sont présents.
Conformité stricte à la spécification	⚠️ Incomplète	environ 90 %	Quelques invariants de course et de récupération ne sont pas fermés.
Fonctionnement nominal			La génération normale continue, se projette et se reconnecte correctement dans le chemin nominal.
fermeture → Activité → réouverture	✅ Correct	environ 96 %	Réservations, époques, retraites, tombstones et claims sont bien implémentés.
Registre et Open backend	✅ Solide	environ 98 %	Direct handoff et preserveLocal sont bons ; la récupération Stop/replayFailed est défectueuse.
Frontend replay/handoff	⚠️ Presque complet	environ 91 %	Le modèle de claim est bon, mais deux fenêtres de course restent ouvertes.
Sécurité destructive	⚠️ À renforcer	environ 86 %	Body Slimbk correctement filtré ; premier-header V1 suivi d'une erreur tardive reste problématique.
Transport et terminal unique	⚠️ Presque complet	environ 92 %	Union stricte, timers, open/closed et Stop exact sont présents.
Activité et Stop détaché	✅ Correct	environ 97 %	

Axe	Verdict	Estimation	Résumé
EasyAgents non-régression	✓ Globalement préservée	environ 94 %	Adaptateur et lifecycle sont propres ; Retry doit être revalidé à la frontière du claim.
Qualité architecturale	✓ Excellente	environ 93 %	Modulaire, typée, documentée et testée, malgré une forte complexité distribuée.
Prêt à être déclaré finalisé	✗ Pas encore	environ 85 %	Les trois courses hautes et les récupérations Stop doivent être corrigées.

6) Conclusion en une phrase

La tranche est très largement livrée et remarquablement architecturée, mais elle ne doit pas être considérée comme finalisée tant que les courses Stop, cleanup sous claim et Retry destructif, ainsi que les lacunes Queue B et transport terminal, ne sont pas corrigées.

7) Quels ensembles de corrections sont nécessaires ?

Lot A — Corrections frontend et Stop 🛠️

Corriger ensemble B1, B4 et B5 :

- libérer `sourceStopPending` lorsqu'une tentative stoppée converge vers `liveSessionReplayFailed` ou devient définitivement indisponible ;
- permettre à Retry de reprendre après un Stop terminé sans handoff ;
- faire retourner à `stop()` un résultat exploitable ou restaurer `transportStopAvailable` lorsque le RPC échoue/refuse ;
- terminer proprement le handshake `abortAckPending` après un échec de Stop ;
- rejeter un item Queue B dont l'identité ne correspond plus à l'identité active **avant** toute mutation du lock, des suffixes ou des buffers différés ;
- ajouter les tests de Stop pré-baseline, échec RPC Stop et arrivée Queue A post-reset.

Lot B — Corrections des opérations destructives 🔒

Corriger ensemble B2 et B3 :

- revalider claim destructif et autorité après chaque frontière asynchrone du cleanup ;
- conserver les marqueurs détachés tant que la revalidation n'autorise pas réellement la suite ;
- refaire le test registre + query manuelle immédiatement avant l'acquisition du claim EasyAgents Retry ;
- garder cette dernière vérification et l'acquisition dans la même pile synchrone ;
- ajouter les courses « nouveau claim pendant cleanup » et « source démarrée pendant le preflight Retry ».

Lot C — Corrections transport et finalisation 🚚

Corriger ensemble B6 et B7 :

- garantir qu'un terminal V1 initial suivi d'un échec transport conserve un terminal visible unique tout en publiant `transportError` ;
- effectuer la correction uniquement dans les sources non générées ;
- intégrer arrêt du son, notification coordinateur et nettoyage final dans l'agrégation ordonnée du handler ;
- ajouter les tests premier `meta.error` puis throw, et transport + erreur coordinateur simultanés.

Lot D — Correction complète 🚩

Appliquer les **trois lots précédents**, compléter les tests ciblés et réaligner les programming docs sur les contrats finaux.

🛠️ Coder les correctifs frontend et Stop

🔒 Coder les correctifs destructifs

🚚 Coder les correctifs transport

🚩 Coder toutes les corrections

</ASSISTANT>

<USER>

[Coder toutes les corrections]

</USER>

<ASSISTANT>

I'm thinking...

</ASSISTANT>

<ASSISTANT>

🔧  scanning git: 🌀 ["État Git avant correction"](#)

🔧  reading project: 🌀 e ["État Stop live Prompt"](#), 🌀 e ["Stop transport frontend"](#), 🌀 e ["Handshake annulation Prompt"](#), 🌀 e ["Entrée Queue B"](#), 🌀 e ["Cleanup panneau détaché"](#), 🌀 e ["Retry EasyAgents claim"](#), 🌀 e ["Finalisation source Prompt"](#), 🌀 e ["Premier header erreur"](#), 📄 [usePromptLiveSession.hook.ts](#), 📄 [useAiStream.hook.ts](#) [L180-L1035], 📄 [usePromptCancelController.hook.ts](#), 📄 [usePromptStreamDispatchGateQueue.hook.ts](#) [L1-L220], 📄 [usePromptStreamDispatchGateQueue.hook.ts](#) [L780-L1025], 📄 [PromptLiveSessionPanelCoordinator.service.ts](#), 📄 [PromptThreadCleanup.manager.ts](#), 📄 [EasyAgentsBatchRunRetry.service.ts](#), 📄 [iaStreamHandlers.ts](#), 📄 [stream.endpoint.ts](#), 📄 [Slimbk.ts](#) [L80-L230], 📄 [callLocalApi.generated.ts](#) [L100-L220], 📄 [callLocalApi.generated.ts](#) [L220-L250], 📄 [Slimbk.ts](#) [L230-L410], 🌀 e ["Terminal V1 puis erreur"](#), 🌀 e ["Erreur après terminal source"](#), 📄 [stream.endpoint.test.ts](#), 📄 [NormalPromptLiveSessionSource.service.test.ts](#) [L320-L506], 📄 [SlimbkResult.type.ts](#), 📄 [iaStreamHandlers.queryDoneSound.test.ts](#) [L1-L180], 📄 [iaStreamHandlers.queryDoneSound.test.ts](#) [L430-L680], 📄 [usePromptLiveSession.normalPrompt.test.tsx](#) [L1-L314], 📄 [useAiStream.hook.promptSourceIdentity.test.ts](#) [L1-L317], 📄 [usePromptCancelController.stopCore.test.tsx](#) [L1-L420], 📄 [PromptStreamDispatchGateQueue.test.ts](#) [L1-L180], 📄 [PromptStreamDispatchGateQueue.test.ts](#) [L1020-L1143], 📄 [PromptLiveSessionPanelCoordinator.service.test.ts](#) [L1-L461], 📄 [EasyAgentsBatchRunRetry.guards.service.test.ts](#) [L1-L340], 📄 [EasyAgentsBatchRunRetry.testHarness.ts](#), 🌀 r ["cleanupOnPromptDispose"](#), 🌀 e ["Coordinateur production"](#), 📄 [openPanel.util.ts](#) [L590-L650], 📄 [AiStreamStartArgs.type.ts](#), 🌀 r ["usePromptCancelCore"](#), 🌀 e ["Stop hook usages"](#), 🌀 e ["Type callback Stop"](#), 📄 [useAiStream.hook.ts](#) [L1-L180], 📄 [usePromptRunController.hook.ts](#) [L560-L650], 📄 [Prompt.tsx](#) [L640-L720], 📄 [PromptInput.tsx](#) [L1030-L1095], 🌀 e ["Construction core annulation"](#), 📄 [usePromptRunController.hook.ts](#) [L420-L475], 🌀 e ["Mocks stop stream"](#), 🌀 e ["Mocks stop frontend"](#), 🌀 e ["Acquisition claim service"](#), 📄 [LiveSessionService.ts](#) [L180-L270], 🌀 e ["Libération claim registre"](#), 📄 [LiveSessionRegistry.service.ts](#) [L820-L900], 🌀 e ["Doc claim cleanup"](#), 🌀 e ["Doc premier header"](#), 🌀 e ["Doc Queue identité"](#), 🌀 e ["Doc Retry claim"](#), 🌀 e ["Doc Stop transport"](#), 📄 [liveSession.programming-doc-readme.md](#) [L1-L160], 📄 [prompt-stream-dispatch-lock.programming-doc-readme.md](#) [L45-L170], 📄 [easyAgents.programming-doc-readme.md](#) [L820-L890], 📄 [easyAgents.programming-doc-readme.md](#) [L890-L940], 📄 [LiveSessionRegistry.service.test.ts](#) [L180-L300], 📄 [NormalPromptLiveSession.adapter.ts](#) [L1-L220], 🌀 e ["Usages latch source Stop"](#), 🌀 e ["Type API live Prompt"](#), 🌀 e ["Reset Queue B"](#), 📄 [usePromptStreamDispatchGateQueue.hook.ts](#) [L220-L300], 📄 [Prompt.autoRunPrompt.test.tsx](#) [L70-L105], 📄 [PromptFallbackPersistBarrier.testHarness.tsx](#) [L185-L220], 🌀 e ["Mocks API live Prompt"](#), 🌀 e ["Mocks stop sans résultat"](#)

🔧  patching file: 📄 [AiStreamStartArgs.type.ts](#)
[AIcode/src/frontend/src/messageBus/AiStreamStartArgs.type.ts](#)

Copy

Review

Accept

PatchFull file

```
- export type AiStreamStartResult = { accepted: false } | AiStreamAcceptedStart;
-
+ export type AiStreamStartResult = { accepted: false } | AiStreamAcceptedStart;
+
+ /** Exact result of one Stop request after the synchronous frontend capability was
+  * claimed. */
+ export type AiStreamStopResult = { accepted: false } | { accepted: true };
+
```

[AIcode/src/frontend/src/messageBus/AiStreamStartArgs.type.ts](#)

Copy

Review

Accept

🔧  patching file: 📄 [useAiStream.hook.ts](#)

[AIcode/src/frontend/src/messageBus/useAiStream.hook.ts](#)

Copy
Review
Accept

PatchFull file

```
-     AiStreamStartResult,
-
+     AiStreamStartResult,
+     AiStreamStopResult,
+


---


-     start: (args: AiStreamStartArgs) => AiStreamStartResult;
-     stop: () => Promise<void>;
-
+     start: (args: AiStreamStartArgs) => AiStreamStartResult;
+     stop: () => Promise<AiStreamStopResult>;
+


---


-     */
-     const stop = useCallback(async () => {
-         try {
-             if (!transportStopAvailableRef.current) return;
-         }
-     });
+     */
+     const stop = useCallback(async (): Promise<AiStreamStopResult> => {
+         // A missing capability means another same-stack Stop already owns the request
+         // or the exact
+         // transport has reached its terminal boundary. This caller must not mutate
+         // that owner's state.
+         if (!transportStopAvailableRef.current) return { accepted: false };
+     });
+


---


-
-         // Consume the exact transport capability synchronously before the first
-         // await. A same-stack
-         // double click therefore cannot send a second source/viewer abort request.
-
+
+         // Capture the exact accepted attempt before consuming the capability. A
+         // refusal may restore
+         // Stop only while this same attempt still owns the hook; a terminal or
+         // successor attempt wins.
+         const currentStream = activeStreamRef.current;
+         const identityAtRequest = activeIdentityRef.current;
+         if (!currentStream || !identityAtRequest) {
+


---


+             updateTransportStopAvailable(false);
+             const currentStream = activeStreamRef.current;
+             const runRef = activeIdentityRef.current?.runRef;
+             if (!currentStream || !runRef) {
+                 throw new Error(
+                     'Accepted stream Stop requires one exact active run identity.',
+                 );
+             }
+
+             updateTransportStopAvailable(false);
+             console.error(
+                 '[useAiStream] accepted Stop capability had no exact active stream
identity.',
+             );
+             return { accepted: false };
+         }
+
+     }
```

```

+      // Consume the exact transport capability synchronously before the first await.
A same-stack
+      // double click therefore cannot send a second source/viewer abort request.
+      updateTransportStopAvailable(false);
+
+      // Accepted Stop deliberately keeps the listener and local stream state
mounted. Terminal
+      // metadata or transport completion remains the single owner of teardown and
outcome publication.
+
+      try {
+        const runRef = identityAtRequest.runRef;
+
+      }
+    } catch (e) {
+
+    }
+    return { accepted: true };
+  } catch (e) {
+
+  } catch (e) {
+    // Non-fatal: backend abort is idempotent and best-effort.
+
+  } catch (e) {
+    // Refusal and transport failure are recoverable while the same attempt is
still active.
+    // Restoring the capability lets the operator retry instead of leaving the
composer in a
+    // permanent abort handshake after a transient RPC failure.
+    if (
+      isSameStreamAttempt(
+        activeIdentityRef.current,
+        identityAtRequest.attemptIdentity,
+      )
+    ) {
+      updateTransportStopAvailable(true);
+    }
+
+    console.error('[useAiStream] abort request failed:', e);
+
+    console.error('[useAiStream] abort request failed:', e);
+    return { accepted: false };
+
+  }
+  // Intentionally no local teardown here to preserve the terminal meta path.
+  // The stream listener will set isStreaming=false through meta/complete.
+
+  }
+

```

[AI Code/src/frontend/src/messageBus/useAiStream.hook.ts](#)

Copy

Review

Accept

  patching file:  [usePromptCancelController.hook.ts](#)

[AI Code/src/frontend/src/app/prompt/Prompt/usePromptCancelController.hook.ts](#)

Error

([what is it?](#))



patching file: [usePromptCancelController.hook.ts](#)

[AI/Code/src/frontend/src/app/prompt/Prompt/usePromptCancelController.hook.ts](#)

Copy

Review

Accept

PatchFull file

```
- import type { AiStreamedChunk } from
'../../../../../sharedlib/types/aiTypes/AiStreamedChunk.type.js';
-
+ import type { AiStreamedChunk } from
'../../../../../sharedlib/types/aiTypes/AiStreamedChunk.type.js';
+ import type { AiStreamStopResult } from
'../../../../../messageBus/AiStreamStartArgs.type.js';
+
-
-      * - When called while streaming, it asks the backend/orchestrator to abort.
-      * - It must not throw in normal operation; errors bubble up to callers/tests.
-
+      * - When called while streaming, it asks the backend/orchestrator to abort.
+      * - A refused or failed request returns `accepted=false` so the UI can release its
+      handshake.
+
-
-      */
-      stop: () => Promise<void>;
-
+      */
+      stop: () => Promise<AiStreamStopResult>;
+
-
-      * Handshake state: true after the user requested cancel of an active stream,
-      * until the stream becomes terminal and `isStreaming` turns false.
-
+      * Handshake state: true after the user requested cancel of an active stream,
+      * until the stream becomes terminal or the exact Stop request is refused.
+
-
-      * - Must preserve the UI-only cancel behavior when no stream is active.
-
+      * - Must preserve the UI-only cancel behavior when no stream is active.
+      * - Resolves true only when cancel was executed; false keeps caller-owned cleanup
+      intact.
+
-
-      */
-      handleStopFromComposer: () => Promise<void>;
-
+      */
+      handleStopFromComposer: () => Promise<boolean>;
+
-
-      * - becomes true once we request abort on an active stream
-      * - becomes false as soon as streaming becomes false (abort acknowledged or
+      terminal stop)
-
+      * - becomes true once we request abort on an active stream
+      * - remains true while the Stop RPC is in flight unless an active stream settles
+      first
+      * - becomes false on refusal, terminal settlement, or an accepted post-render Stop
+      response
+
-
-      const abortAckPendingRef = useRef<boolean>(false);
-

```

```

+     const abortAckPendingRef = useRef<boolean>(false);
+     const stopRequestInFlightRef = useRef<boolean>(false);
+     const stopRequestStartedWhileStreamingRef = useRef<boolean>(false);
+     const isStreamingRef = useRef(isStreaming);
+     isStreamingRef.current = isStreaming;
+
-     isStreamingRef.current = isStreaming;
-
+     isStreamingRef.current = isStreaming;
+
+     /** Release both synchronous and rendered abort-handshake authorities after
terminal or refusal. */
+     const clearAbortHandshake = useCallback(()> void => {
+       abortAckPendingRef.current = false;
+       setAbortAckPending(false);
+     }, []);
+
-     useEffect(() => {
-       if (!isStreaming && !transportStopAvailable) {
-
+     useEffect(() => {
+       if (
+         !isStreaming &&
+         !transportStopAvailable &&
+         (!stopRequestInFlightRef.current ||
+          stopRequestStartedWhileStreamingRef.current)
+       ) {
+
-       // The ref is required because two Stop callbacks can run before React
commits state.
-       abortAckPendingRef.current = false;
-       setAbortAckPending(false);
-
+       // The ref is required because two Stop callbacks can run before React
commits state.
+       clearAbortHandshake();
+
-     }
-     }, [isStreaming, transportStopAvailable]);
-
+     }
+     }, [clearAbortHandshake, isStreaming, transportStopAvailable]);
+
-     */
-     const handleStopFromComposer = useCallback(async (): Promise<void> => {
-
+     */
+     const handleStopFromComposer = useCallback(async (): Promise<boolean> => {
+
-       if (abortAckPendingRef.current) {
-         return;
-
+       if (abortAckPendingRef.current) {
+         return false;
+
-       lastMetaChunkFinalizedRef.current = undefined;
-
-       return;
-

```

```

+         lastMetaChunkFinalizedRef.current = undefined;
+
+         return true;
+
-         // Arm the ref before state so a same-stack second click cannot reach the
transport.
-
+         // Arm the ref before state so a same-stack second click cannot reach the
transport.
+         const startedWhileStreaming = isStreaming;
+
-         abortAckPendingRef.current = true;
-
+         abortAckPendingRef.current = true;
+         stopRequestInFlightRef.current = true;
+         stopRequestStartedWhileStreamingRef.current = startedWhileStreaming;
+
-
-         // Ask the backend to stop streaming (we do not teardown locally; terminal meta
will arrive).
-         await stop();
-
+
+         // Ask the backend to stop streaming without tearing down locally; an accepted
request keeps
+         // the handshake armed until terminal metadata arrives. A refusal restores Stop
immediately.
+         let stopResult: AiStreamStopResult;
+         try {
+             stopResult = await stop();
+         } catch (error) {
+             // Injected adapters/tests may still throw. Never retain a handshake that
has no accepted abort.
+             stopRequestInFlightRef.current = false;
+             stopRequestStartedWhileStreamingRef.current = false;
+             clearAbortHandshake();
+             throw error;
+         }
+         stopRequestInFlightRef.current = false;
+         stopRequestStartedWhileStreamingRef.current = false;
+         if (!stopResult.accepted) {
+             clearAbortHandshake();
+             return false;
+         }
+
-         metaOnlyFinalizedTurnIdRef.current = undefined;
-         lastMetaChunkFinalizedRef.current = undefined;
-
+         metaOnlyFinalizedTurnIdRef.current = undefined;
+         lastMetaChunkFinalizedRef.current = undefined;
+
+         // A post-render transport capability may exist after rendering already became
idle. In that
+         // path there will be no later isStreaming transition to close the handshake,
so the accepted
+         // RPC response itself is the exact acknowledgement boundary.
+         if (!startedWhileStreaming || !isStreamingRef.current) {
+             clearAbortHandshake();
+         }
+         return true;
+

```

```

-         baseAssistantTextRef,
-
+         baseAssistantTextRef,
+         clearAbortHandshake,
+
-
-     /** Core stop handler (UI-only cancel vs transport abort). */
-     handleStopCore: () => Promise<void>;
-
+     /** Core stop handler (UI-only cancel vs transport abort). */
+     handleStopCore: () => Promise<boolean>;
+
-
-         // 2) Stop (UI-only cancel vs transport abort).
-         await handleStopCore();
-
+
+         // 2) Stop (UI-only cancel vs transport abort). Refusal keeps every caller-
+         owned notice intact.
+         const didCancel = await handleStopCore();
+         if (!didCancel) {
+             return;
+         }
+
-     }
-     await handleStopCore();
-
+     }
+     const didCancel = await handleStopCore();
+     if (!didCancel) {
+         return;
+     }
+

```

[AI Code/src/frontend/src/app/prompt/Prompt/usePromptCancelController.hook.ts](#)

Copy

Review

Accept



patching file: [usePromptLiveSession.hook.ts](#)

[AI Code/src/frontend/src/app/prompt/liveSession/core/usePromptLiveSession.hook.ts](#)

Copy

Review

Accept

PatchFull file

```

- export type PromptLiveSessionApi = {
-     markSourceStopRequested: () => boolean;
-
+ export type PromptLiveSessionApi = {
+
-     sourceStopPending: boolean;
-     stopSource: () => Promise<void>;
-
+     sourceStopPending: boolean;
+     stopSource: () => Promise<boolean>;
+
-
-     const [sourceStopPending, setSourceStopPending] = useState(false);
-
+     const [sourceStopPending, setSourceStopPending] = useState(false);
+

```

```

+      /** Release the exact source-Stop UI latch after refusal, terminal handoff, or
+      replay failure. */
+      const clearSourceStopPending = useCallback(): void => {
+        sourceStopPendingRef.current = false;
+        if (mountedRef.current) {
+          setSourceStopPending(false);
+        }
+      }, []);
+
-      'directAfterReplay'
-    ) {
-
+      'directAfterReplay'
+    ) {
+      // A source Stop may retire a non-replayable reservation before Open
+      becomes ready.
+      // Replay failure is a stable surface, so Retry must not remain
+      disabled by the old latch.
+      clearSourceStopPending();
+
-      await closeSelf();
-    },
-    [],
-
+      await closeSelf();
+    },
+    [clearSourceStopPending],
+
-      handoffStartedRef.current = true;
-
+      handoffStartedRef.current = true;
+      clearSourceStopPending();
+
-      params.setHandoffFailureMessage(undefined);
-    }, [params]);
-
+      params.setHandoffFailureMessage(undefined);
+    }, [clearSourceStopPending, params]);
+
-      'The live-session transport completed without terminal data.',
-    );
-
+      'The live-session transport completed without terminal data.',
+    );
+    if (
+      params.bootstrap.presentationAdapter.handoffPolicy ===
+      'directAfterReplay'
+    ) {
+      clearSourceStopPending();
+    }
+
-    }, [params]);
-
-    useEffect(() => {
-
+    }, [clearSourceStopPending, params]);
+
+    useEffect(() => {

```

```

-         'The live-session transport ended with an error.',
-     );
-
+         'The live-session transport ended with an error.',
+     );
+     if (
+         params.bootstrap.presentationAdapter.handoffPolicy ===
+         'directAfterReplay'
+     ) {
+         clearSourceStopPending();
+     }
+

```

```

-     );
-     }, [params]));
-
+     );
+     }, [clearSourceStopPending, params]));
+

```

```

-     /** Stop the exact normal-Prompt source even before a live observer transport has
-     been accepted. */
-     const stopSource = useCallback(async (): Promise<void> => {
-
+     /** Stop the exact normal-Prompt source even before a live observer transport has
+     been accepted. */
+     const stopSource = useCallback(async (): Promise<boolean> => {
+

```

```

-         const bootstrap = paramsRef.current.bootstrap;
-         if (!bootstrap || !markSourceStopRequested()) return;
-
+         const bootstrap = paramsRef.current.bootstrap;
+         if (!bootstrap || !markSourceStopRequested()) return false;
+

```

```

-         if (!result.accepted && mountedRef.current) {
-             sourceStopPendingRef.current = false;
-             setSourceStopPending(false);
-
+         if (!result.accepted && mountedRef.current) {
+             clearSourceStopPending();
+

```

```

-             clearSourceStopPending();
-         }
-
+         clearSourceStopPending();
+     }
+     return result.accepted;
+

```

```

-     } catch (error) {
-         sourceStopPendingRef.current = false;
-         if (mountedRef.current) setSourceStopPending(false);
-
+     } catch (error) {
+         clearSourceStopPending();
+

```

```

-         console.error('[usePromptLiveSession] source Stop failed:', error);
-
+         console.error('[usePromptLiveSession] source Stop failed:', error);
+         return false;
+

```

```

-     }
-     }, [markSourceStopRequested]);
-
+     }
+     }, [clearSourceStopPending, markSourceStopRequested]);
+
-     useEffect(() => {
-         if (params.promptRuntimeMode !== 'normal') return;
-         sourceStopPendingRef.current = false;
-         setSourceStopPending(false);
-     }, [params.promptRuntimeMode]);
-
+     useEffect(() => {
+         // Both successful handoff and stable replay failure end the exact source-Stop
transaction.
+         // The latter must re-enable Retry even when Stop retired the source before a
baseline existed.
+         if (
+             params.promptRuntimeMode !== 'normal' &&
+             params.promptRuntimeMode !== 'liveSessionReplayFailed'
+         )
+             return;
+         clearSourceStopPending();
+     }, [clearSourceStopPending, params.promptRuntimeMode]);
+
-     () => ({
-         markSourceStopRequested,
-
+     () => ({
+
-     [
-         markSourceStopRequested,
-
+     [
+

```

[AI Code/src/frontend/src/app/prompt/liveSession/core/usePromptLiveSession.hook.ts](#)

Copy

Review

Accept

  **patching file:**  [Prompt.tsx](#)

[AI Code/src/frontend/src/app/prompt/Prompt/Prompt.tsx](#)

Copy

Review

Accept

PatchFull file

```

-     const {
-         markSourceStopRequested,
-
+     const {
+
-
-     /** Route the unique composer Stop to the accepted transport or directly to the
exact source reservation. */
-     const handleEffectiveStopCore = useCallback(async (): Promise<void> => {
-         if (transportStopAvailable) {
-             if (sourceStopAvailable && !markSourceStopRequested()) {
-                 return;
-             }
-             await handleStopFromComposer();
-

```

```

-         return;
-     }
-     if (!sourceStopAvailable) {
-         await handleStopFromComposer();
-         return;
-     }
-     await stopSource();
-
+     /** Route the unique composer Stop to the accepted transport or directly to the
+     exact source reservation. */
+     const handleEffectiveStopCore = useCallback(async (): Promise<boolean> => {
+         // A normal-Prompt replay always owns the exact source Stop route, including
+         the period after
+         // its observer listener mounted. Keeping one owner prevents split latches
+         across two controllers.
+         if (sourceStopAvailable) {
+             return await stopSource();
+         }
+
+         // Classic Prompt and EasyAgents viewer transports retain the generic transport
+         abort handshake.
+         return await handleStopFromComposer();
+     });

```

```

-     handleStopFromComposer,
-     markSourceStopRequested,
-
+     handleStopFromComposer,
+

```

```

-     stopSource,
-     transportStopAvailable,
-
+     stopSource,
+

```

[AI Code/src/frontend/src/app/prompt/Prompt/Prompt.tsx](#)

Copy

Review

Accept



patching file: [usePromptLiveSession.normalPrompt.test.tsx](#)

[AI Code/src/frontend/src/app/prompt/liveSession/core/_test_/usePromptLiveSession.normalPrompt.test.tsx](#)

Copy

Review

Accept

PatchFull file

```

-     expect(result.current.api.sourceStopAvailable).toBe(true);
-     let firstStop: Promise<void> | undefined;
-
+     expect(result.current.api.sourceStopAvailable).toBe(true);
+     let firstStop: Promise<boolean> | undefined;
+
-
-     expect(result.current.api.sourceStopPending).toBe(true);
-     expect(result.current.api.markSourceStopRequested()).toBe(false);
-
+     expect(result.current.api.sourceStopPending).toBe(true);
+

```

```

-     it('maps initial not_found to an abortable replay-failed surface', async () => {
-
+

```

```

+     it('releases an accepted pre-baseline Stop latch when replay becomes unavailable
and allows Retry', async () => {
+         let resolveInitialOpen: ((value: unknown) => void) | undefined;
+         let resolveAbort: ((value: unknown) => void) | undefined;
+         callLocalApiMock
+             .mockImplementationOnce(
+                 () =>
+                     new Promise((resolve) => {
+                         resolveInitialOpen = resolve;
+                     }),
+             )
+             .mockImplementationOnce(
+                 () =>
+                     new Promise((resolve) => {
+                         resolveAbort = resolve;
+                     }),
+             )
+             .mockResolvedValueOnce({
+                 kind: 'terminal',
+                 session: {
+                     meta: {
+                         formatVersionNumber: 1,
+                         sessionId: locator.sessionId,
+                         title: 'Chat',
+                         createdAt: 1,
+                         updatedAt: 2,
+                         firstSendState: 'postSend',
+                     },
+                     messages: [],
+                 },
+                 threadId: locator.sessionId,
+                 turnId: bootstrap.runRef.turnId,
+                 sourceId: bootstrap.runRef.sourceId,
+             });
+         const { result } = renderHook(() =>
+             useNormalPromptLiveSessionHarness(),
+         );
+
+         // Let the initial Open claim its in-flight slot before Stop retires the
reservation.
+         await act(async () => {
+             await Promise.resolve();
+         });
+         let stopPromise: Promise<boolean> | undefined;
+         act(() => {
+             stopPromise = result.current.api.stopSource();
+         });
+         await act(async () => {
+             resolveAbort?.({ accepted: true });
+             await stopPromise;
+         });
+         expect(result.current.api.sourceStopPending).toBe(true);
+
+         // The retired source makes the pending Open unavailable. That stable failure
must release
+         // the Stop latch so the operator can execute one preserve-local Retry
immediately.
+         await act(async () => {
+             resolveInitialOpen?.({ kind: 'not_found' });
+             await Promise.resolve();
+             await Promise.resolve();
+         });
+         expect(result.current.mode).toBe('liveSessionReplayFailed');

```

[AICode/src/frontend/src/app/prompt/liveSession/core/ test /usePromptLiveSession.normalPrompt.test.tsx](#)

Copy

Review

Accept

[AICode/src/frontend/src/app/prompt/Prompt/ test /Prompt.autoRunPrompt.test.tsx](#)

Copy

Review

Accept

PatchFull file

[AICode/src/frontend/src/app/prompt/Prompt/ test /Prompt.autoRunPrompt.test.tsx](#)

Copy

Review

Accept

[AICode/src/frontend/src/app/prompt/Prompt/ test /PromptFallbackPersistBarrier.testHarness.tsx](#)

Copy

Review

Accept

PatchFull file

[AICode/src/frontend/src/app/prompt/Prompt/ test /PromptFallbackPersistBarrier.testHarness.tsx](#)

Copy
Review
Accept



patching file: [useAiStream.hook.promptSourceIdentity.test.ts](#)

[AI Code/src/frontend/src/messageBus/_test_/useAiStream.hook.promptSourceIdentity.test.ts](#)

Copy
Review
Accept

PatchFull file

```
-      AiStreamStartResult,  
-  
+      AiStreamStartResult,  
+      AiStreamStopResult,  
+  
  
-      let startResult: AiStreamStartResult | undefined;  
-      let firstStop: Promise<void> | undefined;  
-  
+      let startResult: AiStreamStartResult | undefined;  
+      let firstStop: Promise<AiStreamStopResult> | undefined;  
+  
  
-      await act(async () => {  
-        await firstStop;  
-  
+      await act(async () => {  
+        await expect(firstStop).resolves.toEqual({ accepted: true });  
+  
  
-      expect(harnessApiRef.current?.transportStopAvailable).toBe(false);  
-    });  
-  
+      expect(harnessApiRef.current?.transportStopAvailable).toBe(false);  
+    });  
+  
+    it('restores the exact Stop capability after refusal or RPC failure while the  
attempt remains active', async () => {  
+      await act(async () => {  
+        root.render(React.createElement(Harness));  
+      });  
+      const api = harnessApiRef.current;  
+      if (!api) throw new Error('HarnessApi not initialized');  
+  
+      let startResult: AiStreamStartResult | undefined;  
+      act(() => {  
+        startResult = api.start({  
+          kind: 'promptQuery',  
+          turnId: 'turn-retry-stop',  
+          sourceId: 'source-retry-stop',  
+          storageScopeRelPath: '',  
+          aiQuery: { type: 'prompt', promptValue: 'Keep running' },  
+          aiQueryOptions: {},  
+        });  
+      });  
+      if (!startResult?.accepted) {  
+        throw new Error('Expected one accepted prompt source.');+      }  
+  
+      callLocalApiMock.mockResolvedValueOnce({ accepted: false });  
+      await act(async () => {  
+        await expect(api.stop()).resolves.toEqual({ accepted: false });
```

```

+    });
+    expect(harnessApiRef.current?.transportStopAvailable).toBe(true);
+
+    callLocalApiMock.mockRejectedValueOnce(
+      new Error('temporary abort transport failure'),
+    );
+    await act(async () => {
+      await expect(api.stop()).resolves.toEqual({ accepted: false });
+    });
+    expect(harnessApiRef.current?.transportStopAvailable).toBe(true);
+
+    await act(async () => {
+      await expect(api.stop()).resolves.toEqual({ accepted: true });
+    });
+    expect(callLocalApiMock).toHaveBeenCalledTimes(3);
+    expect(harnessApiRef.current?.transportStopAvailable).toBe(false);
+  });
+

```

[AI Code/src/frontend/src/messageBus/ test /useAiStream.hook.promptSourceIdentity.test.ts](#)

Copy

Review

Accept

  patching file:  [usePromptCancelController.stopCore.test.tsx](#)

[AI Code/src/frontend/src/app/prompt/Prompt/ test /usePromptCancelController.stopCore.test.tsx](#)

Error

([what is it?](#))

  patching file:  [usePromptCancelController.stopCore.test.tsx](#)

[AI Code/src/frontend/src/app/prompt/Prompt/ test /usePromptCancelController.stopCore.test.tsx](#)

Copy

Review

Accept

PatchFull file

```

- import type { AiStreamedChunk } from
-   '.../sharedlib/types/aiTypes/AiStreamedChunk.type.js';
+ import type { AiStreamedChunk } from
+   '.../sharedlib/types/aiTypes/AiStreamedChunk.type.js';
+ import type { AiStreamStopResult } from
+   '.../messageBus/AiStreamStartArgs.type.js';
+

```

```

-   stopEnabled: boolean;
-   handleStopFromComposer: () => Promise<void>;
+   stopEnabled: boolean;
+   handleStopFromComposer: () => Promise<boolean>;
+

```

```

-   transportStopAvailable?: boolean;
-   stop: () => Promise<void>;
+   transportStopAvailable?: boolean;
+   stop: () => Promise<AiStreamStopResult>;
+

```

```

- test('usePromptCancelCore: UI-only cancel does not call stop() and resets invariants',
-   async () => {
-     const host = document.createElement('div');
-     const root = createRoot(host);

```

```

-
-   const stop = jest.fn<() => Promise<void>>().mockResolvedValue(undefined);
-
+ test('usePromptCancelCore: UI-only cancel does not call stop() and resets invariants',
+   async () => {
+     const host = document.createElement('div');
+     const root = createRoot(host);
+
+     const stop = jest
+       .fn<() => Promise<AiStreamStopResult>>()
+       .mockResolvedValue({ accepted: true });
+


---


-
-   const stop = jest.fn<() => Promise<void>>().mockResolvedValue(undefined);
-
+
+   const stop = jest
+     .fn<() => Promise<AiStreamStopResult>>()
+     .mockResolvedValue({ accepted: true });
+


---


-   const root = createRoot(host);
-   let resolveStop: (() => void) | undefined;
-   const stop = jest.fn<() => Promise<void>>(
-
+   const root = createRoot(host);
+   let resolveStop: (value: AiStreamStopResult) => void | undefined;
+   const stop = jest.fn<() => Promise<AiStreamStopResult>>(
+


---


-     async () =>
-       await new Promise<void>((resolve) => {
-
+     async () =>
+       await new Promise<AiStreamStopResult>((resolve) => {
+


---


-
-   let firstStop: Promise<void> | undefined;
-
+
+   let firstStop: Promise<boolean> | undefined;
+


---


-
-   resolveStop?.();
-
+
+   resolveStop?.({ accepted: true });
+


---


-   // Stop therefore performs the normal post-abort pending cleanup exactly once.
-   expect(setUiPending).toHaveBeenCalledWith(false);
-
+   // Stop therefore performs the normal post-abort pending cleanup exactly once.
+   expect(setUiPending).toHaveBeenCalledWith(false);
+   expect(snapshotRef.current?.abortAckPending).toBe(false);
+   expect(snapshotRef.current?.stopEnabled).toBe(true);
+


---


-   expect(snapshotRef.current?.stopEnabled).toBe(true);
-
-   await act(async () => {
-     root.unmount();
-   });

```

```

- });
-
+ expect(snapshotRef.current?.stopEnabled).toBe(true);
+
+ await act(async () => {
+   root.unmount();
+ });
+ });
+
+ test('usePromptCancelCore: refused transport Stop releases the handshake without
clearing caller state', async () => {
+   const host = document.createElement('div');
+   const root = createRoot(host);
+   const stop = jest
+     .fn<() => Promise<AiStreamStopResult>>()
+     .mockResolvedValue({ accepted: false });
+   const setUiPending = jest.fn<(v: SetStateAction<boolean>) => void>();
+   const snapshotRef: { current: CancelCoreApiSnapshot | undefined } = {
+     current: undefined,
+   };
+
+   await act(async () => {
+     root.render(
+       <Harness
+         isStreaming={true}
+         transportStopAvailable={true}
+         stop={stop}
+         setUiPending={setUiPending}
+         snapshotRef={snapshotRef}
+       />,
+     );
+   });
+   const api = snapshotRef.current;
+   if (!api) throw new Error('Expected cancel core API.');
```

[AICode/src/frontend/src/app/prompt/Prompt/ _test_/usePromptCancelController.stopCore.test.tsx](#)

Copy

Review

Accept



reading project: [usePromptStreamDispatchGateQueue.hook.ts](#) [L700-L810]

  patching file:  [usePromptStreamDispatchGateQueue.hook.ts](#)

[AICode/src/frontend/src/messageBus/usePromptStreamDispatchGateQueue.hook.ts](#)

Error

([what is it?](#))

  patching file:  [PromptStreamDispatchGateQueue.test.ts](#)

[AICode/src/frontend/src/messageBus/ test /PromptStreamDispatchGateQueue.test.ts](#)

Copy

Review

Accept

PatchFull file

```
-      let gate: PromptStreamDispatchGateQueueApi | undefined;
-
+      let gate: PromptStreamDispatchGateQueueApi | undefined;
+      let activeAttempt: AiStreamAttemptIdentity = ATTEMPT_A;
+
-
-      dispatchMerged,
-      getAttemptIdentity: getAttemptA,
-
+      dispatchMerged,
+      getAttemptIdentity: () => activeAttempt,
+
-
-      queue.resetInternalState();
-
+      activeAttempt = ATTEMPT_B;
+      queue.resetInternalState();
+
-
-    });
-  });
-
+  });
+
+  it('drops stale Queue A ingress before tool-call gating can lock the successor
attempt', async () => {
+    PromptStreamLockStore.reset();
+    const container = document.createElement('div');
+    document.body.appendChild(container);
+    const root = createRoot(container);
+    const dispatchCalls: AiStreamedChunk[] = [];
+    let gate: PromptStreamDispatchGateQueueApi | undefined;
+    let activeAttempt: AiStreamAttemptIdentity = ATTEMPT_A;
+
+    function StaleIngressHarness(): JSX.Element | null {
+      const dispatchMerged = useCallback(
+        async (chunk: AiStreamedChunk): Promise<void> => {
+          dispatchCalls.push(chunk);
+        },
+        [],
+      );
+      const queueOptions = useMemo(
+        () => ({
+          dispatchMerged,
+          getAttemptIdentity: () => activeAttempt,
+        }),
+        [dispatchMerged],
+      );
+      const api = usePromptStreamDispatchGateQueue(queueOptions);
```

```

+         useEffect(() => {
+             gate = api;
+         }, [api]);
+         return null;
+     }
+
+     await act(async () => {
+         root.render(React.createElement(StaleIngressHarness));
+     });
+     const queue = gate;
+     if (!queue) throw new Error('Gate queue API not initialized');
+
+     activeAttempt = ATTEMPT_B;
+     PromptStreamLockStore.reset();
+     queue.resetInternalState();
+     const staleCodeframe: AiStreamedToolCall = {
+         callId: 'stale-codeframe',
+         toolName: AiToolName.file_write_patch,
+         kind: 'static',
+         args: [],
+         isParsingComplete: false,
+     };
+
+     await act(async () => {
+         await queue.enqueueMergedForAttempt(
+             toolCallsChunk([staleCodeframe]),
+             ATTEMPT_A,
+         );
+         await queue.whenIdle();
+     });
+
+     expect(dispatchCalls).toEqual([]);
+     expect(PromptStreamLockStore.getLockedCallId()).toBeNull();
+
+     await act(async () => {
+         await queue.enqueueMergedForAttempt(
+             tokenChunk('successor'),
+             ATTEMPT_B,
+         );
+         await queue.whenIdle();
+     });
+     expect(dispatchCalls).toEqual([tokenChunk('successor')]);
+
+     await act(async () => {
+         root.unmount();
+     });
+     document.body.removeChild(container);
+ });
+ });
+

```

[AI/Code/src/frontend/src/messageBus/_test_/PromptStreamDispatchGateQueue.test.ts](#)

Copy

Review

Accept



patching file: [usePromptStreamDispatchGateQueue.hook.ts](#)

[AI/Code/src/frontend/src/messageBus/usePromptStreamDispatchGateQueue.hook.ts](#)

Error

([what is it?](#))

  patching file:  [usePromptStreamDispatchGateQueue.hook.ts](#)

[AI Code/src/frontend/src/messageBus/usePromptStreamDispatchGateQueue.hook.ts](#)

Copy

Review

Accept

PatchFull file

```
-      Array.isArray(chunk.value.value)
-    );
-  }
-
+      Array.isArray(chunk.value.value)
+    );
+  }
+
+ /** Compare one queued attempt with the current synchronous stream authority. */
+ function isSameAttemptIdentity(
+   current: AiStreamAttemptIdentity | undefined,
+   queued: AiStreamAttemptIdentity,
+ ): boolean {
+   return (
+     current?.sourceId === queued.sourceId &&
+     current.streamInstanceId === queued.streamInstanceId
+   );
+ }
+
-       if (pending.generation !== currentGeneration) return true;
-
+       if (pending.generation !== currentGeneration) return true;
+       if (
+         !isSameAttemptIdentity(
+           opts.getAttemptIdentity(),
+           pending.attemptIdentity,
+         )
+       ) {
+         // A suffix captured by a superseded attempt must disappear before it
can relock
+         // PromptStreamLockStore or mutate any successor Queue B accumulator.
+         return true;
+       }
+
-       deferredWhileLockedRef.current.shift();
-       return true;
-     }
-
+       deferredWhileLockedRef.current.shift();
+       return true;
+     }
+     if (
+       !isSameAttemptIdentity(
+         opts.getAttemptIdentity(),
+         deferred.attemptIdentity,
+       )
+     ) {
+       deferredWhileLockedRef.current.shift();
+       return true;
+     }
+
-   }
-
-   const lockedCallId = lockedCallIdRef.current;
```

```

+         }
+         if (
+             !isSameAttemptIdentity(
+                 opts.getAttemptIdentity(),
+                 head.attemptIdentity,
+             )
+         ) {
+             // Queue A may release an old merged item after the successor reset already
+             // completed.
+             // Drop it before gating so stale tool_calls cannot acquire the successor's
+             // global lock.
+             queueRef.current.shift();
+             head.resolve();
+             return true;
+         }
+
+         const lockedCallId = lockedCallIdRef.current;
+
+         ): Promise<void> => {
+
+         ): Promise<void> => {
+             // Reject stale Queue A ingress before it enters Queue B. Resolving rather
+             // than rejecting
+             // intentionally releases upstream backpressure for work that no longer
+             // owns UI state.
+             if (
+                 !isSameAttemptIdentity(
+                     opts.getAttemptIdentity(),
+                     attemptIdentity,
+                 )
+             ) {
+                 return Promise.resolve();
+             }
+
+             });
+         },
+         [scheduleDrain],
+
+         });
+     },
+     [opts, scheduleDrain],
+

```

[AICode/src/frontend/src/messageBus/usePromptStreamDispatchGateQueue.hook.ts](#)

Copy

Review

Accept

  **patching file:**  [LiveSessionRegistry.service.ts](#)

[AICode/src/extension/src/aiTools/orch/liveSession/registry/LiveSessionRegistry.service.ts](#)

Copy

Review

Accept

PatchFull file

```

-         this.destructionClaimByLocatorKey.delete(locatorKey);
-
+         this.destructionClaimByLocatorKey.delete(locatorKey);
+         // Retire the reverse lookup together with the locator entry. Cleanup
+         // coordination now
+         // acquires short-lived claims too, so retaining released handles would leak
+         // one entry per close.

```

```
+         this.destructionLocatorKeyByClaim.delete(claim);
```

```
+ 
```

[AICode/src/extension/src/aiTools/orch/liveSession/registry/LiveSessionRegistry.service.ts](#)

Copy

Review

Accept



patching file: [PromptLiveSessionPanelCoordinator.service.ts](#)

[AICode/src/extension/src/extension/panels/prompt/liveSession/PromptLiveSessionPanelCoordinator.service.ts](#)

Copy

Review

Accept

PatchFull file

```
-     private readonly detachedLocatorKeys = new Set<string>();
```

```
- 
```

```
+     private readonly detachedLocatorKeys = new Set<string>();
```

```
+     private readonly cleanupReleaseWakeUpSuppressionKeys = new Set<string>();
```

```
+ 
```

```
-         | 'releaseViewerLease'
-         | 'isLocatorDestructionClaimed'
```

```
- 
```

```
+         | 'releaseViewerLease'
+         | 'acquireLocatorDestructionClaim'
+         | 'releaseLocatorDestructionClaim'
```

```
+ 
```

```
-         this.deps.liveSession.onDestructionClaimReleased((event) => {
```

```
- 
```

```
+         this.deps.liveSession.onDestructionClaimReleased((event) => {
```

```
+             const key = this.locatorKey(event.locator);
```

```
+             if (this.cleanupReleaseWakeUpSuppressionKeys.delete(key)) {
```

```
+                 // The coordinator's own short-lived cleanup claim must not recursively
```

```
schedule the
```

```
+                 // same failed cleanup forever. External Delete/Retry releases still
```

```
trigger retries.
```

```
+                 return;
```

```
+             }
```

```
+ 
```

```
-     }
```

```
- 
```

```
-     if (
-         this.findActiveAuthority(locator) ||
-         this.deps.liveSession.isLocatorDestructionClaimed(locator)
-     ) {
```

```
- 
```

```
+     }
```

```
+     if (this.findActiveAuthority(locator)) {
+         this.detachedLocatorKeys.add(key);
+         return;
```

```
+     }
```

```
+ 
```

```
+     // Acquire the same locator-wide mutex used by durable Delete and EasyAgents
```

```
Retry. The
```

```
+     // synchronous claim closes the TOCTOU window before the first cleanup await:
no new source,
```

```
+     // Open, or competing destructive operation can start until every cleanup step
converges.
```

```
+     const cleanupClaim =
```

```
+         this.deps.liveSession.acquireLocatorDestructionClaim(locator);
```

```
+     if (!cleanupClaim) {
```

```
+ 
```

```

-     }
-     const descriptor = this.descriptorByLocator.get(key);
-
-     }
-
-     const errors: Error[] = [];
-     try {
-         const descriptor = this.descriptorByLocator.get(key);
-
-         // Keep both detached and descriptor markers throughout asynchronous
cleanup. If one step fails,
-         // a later external observer/claim transition can retry the exact cleanup
without losing identity.
-         detachPromptSession(locator);
-         await cleanupOnPromptDispose(locator);
-         if (descriptor) {
-             // Exact sound state survives replay attempts and is retired only after
all live authority disappeared.
-             cleanupPromptQueryDoneSoundRun({
-                 locator,
-                 turnId: descriptor.liveSessionTurnId,
-                 sourceId: descriptor.liveSessionSourceId,
-             });
-         }
-         cleanupOnPromptQueryDoneSoundDispose(locator.sessionId);
-
-
-         // Keep both detached and descriptor markers throughout asynchronous cleanup.
If one step fails,
-         // a later observer/claim wake-up can retry the exact cleanup instead of losing
its identity.
-         detachPromptSession(locator);
-         await cleanupOnPromptDispose(locator);
-         if (descriptor) {
-             // Exact sound state survives replay attempts and is retired only after all
live authority disappeared.
-             cleanupPromptQueryDoneSoundRun({
-                 locator,
-                 turnId: descriptor.liveSessionTurnId,
-                 sourceId: descriptor.liveSessionSourceId,
-             });
-
-
-         // Trivial chat cleanup and Sidebar removal execute under the same exact
claim, so an external
+         // delete/retry cannot overtake the awaited disk operations and race the
detached mapping.
+         const persistence = new ChatPersistence();
+         const result =
+             await
persistence.trivialUntitledCleanup.cleanupAfterPromptCloseBestEffort(
+                 locator,
+             );
+         if (
+             result.action === 'not_persisted' ||
+             result.action === 'deleted_disk'
+         ) {
+             await updateSidebarStore((store) => {
+                 const items = store.sidebarContentState.chatState.items;
+                 const index = items.findIndex(
+                     (item) =>
+                         item.sessionId === locator.sessionId &&
+                         item.storageScopeRelPath ===

```

```

+
+         locator.storageScopeRelPath,
+
+         );
+         if (index >= 0) items.splice(index, 1);
+     });
+ }
+ this.detachedLocatorKeys.delete(key);
+ this.descriptorByLocator.delete(key);
+ } catch (error) {
+     errors.push(normalizeCoordinatorError(error));
+
+ }
+
+ cleanupOnPromptQueryDoneSoundDispose(locator.sessionId);
+
+ }
+
+
+ // Trivial chat cleanup is destructive and therefore runs only after all live
authority disappeared.
+ const persistence = new ChatPersistence();
+ const result =
+     await persistence.trivialUntitledCleanup.cleanupAfterPromptCloseBestEffort(
+         locator,
+     );
+ if (
+     result.action === 'not_persisted' ||
+     result.action === 'deleted_disk'
+ ) {
+     await updateSidebarStore((store) => {
+         const items = store.sidebarContentState.chatState.items;
+         const index = items.findIndex(
+             (item) =>
+                 item.sessionId === locator.sessionId &&
+                 item.storageScopeRelPath ===
+                     locator.storageScopeRelPath,
+         );
+
+         // Release in every outcome. Suppress only this exact synchronous release event
so a persistent
+         // cleanup failure cannot create an unbounded self-retry loop; external
releases still wake it.
+         this.cleanupReleaseWakeUpSuppressionKeys.add(key);
+         try {
+             if (
+                 !this.deps.liveSession.releaseLocatorDestructionClaim(
+                     cleanupClaim,
+                 )
+             ) {
+                 errors.push(
+                     new Error(
+                         'Prompt detached cleanup lost its exact live-session
destruction claim.',
+                     ),
+                 );
+
+             };
+             if (index >= 0) items.splice(index, 1);
+         });
+
+     });
+ }
+ } catch (error) {
+     errors.push(normalizeCoordinatorError(error));
+ } finally {

```

```

+         this.cleanupReleaseWakeUpSuppressionKeys.delete(key);
+
-     }
-     this.detachedLocatorKeys.delete(key);
-     this.descriptorByLocator.delete(key);
-
+     }
+     throwCoordinatorErrors(
+         errors,
+         'Prompt detached cleanup completed with ordered failures.',
+     );
+

```

[AI Code/src/extension/src/extension/panels/prompt/liveSession/PromptLiveSessionPanelCoordinator.service.ts](#)

Copy

Review

Accept

  **patching file:**  [PromptLiveSessionPanelCoordinator.service.test.ts](#)

[AI Code/src/extension/src/extension/panels/prompt/liveSession/ test /](#)

[PromptLiveSessionPanelCoordinator.service.test.ts](#)

Copy

Review

Accept

PatchFull file

```

-     LiveSessionDestructionClaimReleasedEvent,
-
+     LiveSessionDestructionClaimReleasedEvent,
+     LiveSessionLocatorDestructionClaim,
+
- const RUN = { locator: LOCATOR, turnId: 'turn', sourceId: 'source' } as const;
-
+ const RUN = { locator: LOCATOR, turnId: 'turn', sourceId: 'source' } as const;
+ const CLEANUP_CLAIM: LiveSessionLocatorDestructionClaim = Object.freeze({
+     token: Symbol('coordinator-cleanup-claim'),
+ });
+
-     jest.fn<LiveSessionService['releaseViewerLease']>();
-     const isLocatorDestructionClaimedMock =
-     jest.fn<LiveSessionService['isLocatorDestructionClaimed']>();
-
+     jest.fn<LiveSessionService['releaseViewerLease']>();
+ const acquireLocatorDestructionClaimMock =
+     jest.fn<LiveSessionService['acquireLocatorDestructionClaim']>();
+ const releaseLocatorDestructionClaimMock =
+     jest.fn<LiveSessionService['releaseLocatorDestructionClaim']>();
+
-         releaseViewerLease: releaseViewerLeaseMock,
-         isLocatorDestructionClaimed: isLocatorDestructionClaimedMock,
-
+         releaseViewerLease: releaseViewerLeaseMock,
+         acquireLocatorDestructionClaim:
+             acquireLocatorDestructionClaimMock,
+         releaseLocatorDestructionClaim:
+             releaseLocatorDestructionClaimMock,
+
-         releaseViewerLeaseMock.mockReturnValue('group');
-         isLocatorDestructionClaimedMock.mockReset();
-         isLocatorDestructionClaimedMock.mockReturnValue(false);

```

```

-
+ releaseViewerLeaseMock.mockReturnValue('group');
+ acquireLocatorDestructionClaimMock.mockReset();
+ acquireLocatorDestructionClaimMock.mockReturnValue(CLEANUP_CLAIM);
+ releaseLocatorDestructionClaimMock.mockReset();
+ releaseLocatorDestructionClaimMock.mockReturnValue(true);
+
-
- activeSnapshots = [];
- isLocatorDestructionClaimedMock.mockReturnValue(true);
-
+ activeSnapshots = [];
+ acquireLocatorDestructionClaimMock.mockReturnValue(undefined);
+
-
- isLocatorDestructionClaimedMock.mockReturnValue(false);
-
+ acquireLocatorDestructionClaimMock.mockReturnValue(CLEANUP_CLAIM);
+
-
- expect(cleanupOnPromptDisposeMock).toHaveBeenCalledTimes(1);
-
+ expect(cleanupOnPromptDisposeMock).toHaveBeenCalledTimes(1);
+ expect(releaseLocatorDestructionClaimMock).toHaveBeenCalledWith(
+   CLEANUP_CLAIM,
+ );
+
-
- it.each([
-
+   it('holds one exact destruction claim until every asynchronous detached cleanup
step settles', async () => {
+     let resolveThreadCleanup: (() => void) | undefined;
+     cleanupOnPromptDisposeMock.mockImplementationOnce(
+       async () =>
+         await new Promise<void>((resolve) => {
+           resolveThreadCleanup = resolve;
+         }),
+     );
+     const coordinator = createCoordinator();
+
+     const disposal = coordinator.onPanelDisposed(LOCATOR);
+     await Promise.resolve();
+     await Promise.resolve();
+
+     expect(acquireLocatorDestructionClaimMock).toHaveBeenCalledWith(
+       LOCATOR,
+     );
+     expect(cleanupOnPromptDisposeMock).toHaveBeenCalledWith(LOCATOR);
+     expect(releaseLocatorDestructionClaimMock).not.toHaveBeenCalled();
+
+     const releaseThreadCleanup = resolveThreadCleanup;
+     if (!releaseThreadCleanup) {
+       throw new Error('Expected detached thread cleanup barrier.');
```

```

+         expect(releaseLocatorDestructionClaimMock).toHaveBeenCalledTimes(1);
+     });
+
+     it('suppresses the coordinator cleanup claim own release wake-up to avoid recursive
retries', async () => {
+         const coordinator = createCoordinator();
+         releaseLocatorDestructionClaimMock.mockImplementationOnce(() => {
+             destructionReleasedListener?.({
+                 locator: LOCATOR,
+                 destructionGeneration: 8,
+             });
+             return true;
+         });
+
+         await coordinator.onPanelDisposed(LOCATOR);
+         await flushBackgroundQueue();
+
+         expect(cleanupOnPromptDisposeMock).toHaveBeenCalledTimes(1);
+         expect(acquireLocatorDestructionClaimMock).toHaveBeenCalledTimes(1);
+         expect(releaseLocatorDestructionClaimMock).toHaveBeenCalledTimes(1);
+         expect(reportBackgroundErrorMock).not.toHaveBeenCalled();
+     });
+
+     it.each([
+

```

[AICode/src/extension/src/extension/panels/prompt/liveSession/_test_/PromptLiveSessionPanelCoordinator.service.test.ts](#)

Copy

Review

Accept



patching file: [EasyAgentsBatchRunRetry.service.ts](#)

[AICode/src/extension/src/aiTools/easyAgents/batch/run/EasyAgentsBatchRunRetry.service.ts](#)

Copy

Review

Accept

PatchFull file

```

-
-         if (
-             this.liveSessionAdapter.hasActiveSourceForLocator(
-                 targetRow.childSessionLocator,
-             ) ||
-             this.isManualQueryRunningForLocator(
-                 targetRow.childSessionLocator,
-             )
-         ) {
-             throw new EasyAgentsError({
-                 code: EASY_AGENT_ERROR_CODES.SYSTEM_ERROR,
-                 message:
-                     EASY_AGENTS_BATCH_RETRY_CHILD_RUNNING_BLOCKED_REASON,
-             });
-         }
-
+
+         this.assertChildDestructionAvailable(
+             targetRow.childSessionLocator,
+         );
+

```

```

-          // Acquire destructive authority immediately before panel close and retain
it through eviction and delete.
-
+
+          // Revalidate runtime authority immediately before the synchronous claim
acquisition. Git and
+          // preparation awaited above, so the earlier precheck cannot authorize this
destructive boundary.
+          // No await is permitted between this guard and the claim: the claim then
blocks every new source.
+          this.assertChildDestructionAvailable(
+              targetRow.childSessionLocator,
+          );
+
+          // Acquire destructive authority immediately before panel close and retain
it through eviction and delete.
+
-      }
-  }
-
+  }
+  }
+
+  /** Refuse destructive Retry whenever the registry or Prompt runtime still owns the
child locator. */
+  private assertChildDestructionAvailable(
+      locator: ChatSessionLocator,
+  ): void {
+      // Both sources are synchronous authorities. Keeping this check synchronous is
what lets the
+      // caller place it directly adjacent to locator-claim acquisition without a
TOCTOU await gap.
+      if (
+          this.liveSessionAdapter.hasActiveSourceForLocator(locator) ||
+          this.isManualQueryRunningForLocator(locator)
+      ) {
+          throw new EasyAgentsError({
+              code: EASY_AGENT_ERROR_CODES.SYSTEM_ERROR,
+              message: EASY_AGENTS_BATCH_RETRY_CHILD_RUNNING_BLOCKED_REASON,
+          });
+      }
+  }
+
+

```

[AI Code/src/extension/src/aiTools/easyAgents/batch/run/EasyAgentsBatchRunRetry.service.ts](#)

Copy

Review

Accept



patching file: [EasyAgentsBatchRunRetry.guards.service.test.ts](#)

[AI Code/src/extension/src/aiTools/easyAgents/batch/run/ test /EasyAgentsBatchRunRetry.guards.service.test.ts](#)

Copy

Review

Accept

PatchFull file

```

-          expect(deleteChatByLocatorMock).not.toHaveBeenCalled();
-      });
-
+          expect(deleteChatByLocatorMock).not.toHaveBeenCalled();
+      });
+
+

```

```

+     it('rechecks registry authority after asynchronous preflight and immediately before
claim acquisition', async () => {
+         const {
+             service,
+             hasActiveSourceForLocatorMock,
+             acquireLocatorDestructionClaimMock,
+             closePromptPanelForLocatorMock,
+             deleteChatByLocatorMock,
+             resolveAuthoritativeStateMock,
+         } = createRetryHarness();
+         resolveAuthoritativeStateMock.mockResolvedValue(
+             buildColdState({
+                 state: 'done',
+                 lineStatus: 'error',
+                 updatedAtMs: 10,
+                 includeChildLocator: true,
+             }),
+         );
+         hasActiveSourceForLocatorMock
+             .mockReturnValueOnce(false)
+             .mockReturnValueOnce(true);
+
+         // The source appears while Git/preparation awaits are in progress. The second
synchronous
+         // guard must reject before the destructive claim, panel close, eviction, or
disk delete.
+         await expect(
+             service.retryLine({
+                 batchParentLocator: PARENT_LOCATOR,
+                 lineNumber: 2,
+             }),
+         ).rejects.toThrow(EASY_AGENTS_BATCH_RETRY_CHILD_RUNNING_BLOCKED_REASON);
+
+         expect(hasActiveSourceForLocatorMock).toHaveBeenCalledTimes(2);
+         expect(acquireLocatorDestructionClaimMock).not.toHaveBeenCalled();
+         expect(closePromptPanelForLocatorMock).not.toHaveBeenCalled();
+         expect(deleteChatByLocatorMock).not.toHaveBeenCalled();
+     });
+
+

```

[AI Code/src/extension/src/aiTools/easyAgents/batch/run/ test /EasyAgentsBatchRunRetry.guards.service.test.ts](#)

Copy

Review

Accept

  **patching file:**  [stream.endpoint.ts](#)

[AI Code/src/extension/src/slimbk-local/endpoints/ai/orch/query/ask/stream.endpoint.ts](#)

Copy

Review

Accept

PatchFull file

```

-     }
- }
-
+     }
+ }
+
+ /** Narrow one durable source terminal that is eligible for the private first-header V1
encoding. */
+ type SourceErrorChunk = Extract<
+     AiStreamedChunk,
+     { type: 'meta'; value: { metaType: 'error' } }
+

```

```

+ >;
+
+ /** Identify the exact terminal shape without weakening the shared AiStreamedChunk
contract. */
+ function isSourceErrorChunk(chunk: AiStreamedChunk): chunk is SourceErrorChunk {
+     return chunk.type === 'meta' && chunk.value.metaType === 'error';
+ }
+
-         let headerSent = false;
-         let funStreamChunk: FunStreamChunk | undefined = undefined;
-
+         let headerSent = false;
+         let funStreamChunk: FunStreamChunk | undefined;
+
-
-         // Only the first durable source error is encoded in the private
non-200 header.
-
+
-         headerSent = true;
-         if (
-             chunk.type === 'meta' &&
-             chunk.value.metaType === 'error'
-         ) {
-             funSendStreamResult(
-
+             headerSent = true;
+             if (isSourceErrorChunk(chunk)) {
+                 // The V1 error is delivered immediately through the header,
while its explicit
+                 // streaming marker keeps the body channel available for any
later transport envelope.
+                 funStreamChunk = funSendStreamResult(
+
-
-         // A terminal error header cannot be followed by an unrelated body
chunk.
-
+
-         throw new Error(
-             'Normal Prompt source emitted data after its terminal error
header.',
-
+         throw new Error(
+             'Normal Prompt source stream writer was not initialized.',
+

```

[AICode/src/extension/src/slimbk-local/endpoints/ai/orch/query/ask/stream.endpoint.ts](#)

Copy

Review

Accept



patching file: [stream.endpoint.test.ts](#)

[AICode/src/extension/src/slimbk-local/endpoints/ai/orch/query/ask/ test /stream.endpoint.test.ts](#)

Copy

Review

Accept

PatchFull file

-

```

-    it('keeps a later meta.error as an ordinary body chunk after the 200 header', async
() => {
-
+
+    it('keeps a first meta.error V1 header streaming when transport fails afterward',
async () => {
+        loadAiCatalogMock.mockResolvedValue({ engines: [] });
+        const firstError = JSON.stringify({
+            turnId: 'turn-1',
+            type: 'meta',
+            value: {
+                metaType: 'error',
+                value: {
+                    reason: 'system_error',
+                    message: 'system error',
+                    details: 'provider terminal',
+                },
+            },
+        });
+        askStreamMock.mockReturnValue(
+            Readable.from(
+                (async function* () {
+                    yield firstError;
+                    throw new Error('transport failed after source terminal');
+                })(),
+            ),
+        );
+        const chunks: string[] = [];
+        const send = jest.fn<(result: SlimbkResult) => (chunk: string) => void>(
+            () => (chunk) => chunks.push(chunk),
+        );
+        const parsedArgs = endpoint.argsSchema.parse(
+            buildArgs({ type: 'prompt', promptValue: 'Hello' }),
+        );
+        stageReservation(parsedArgs);
+
+        await expect(
+            endpoint.streamedHandler({ headers: {}, args: parsedArgs }, send),
+        ).rejects.toThrow('transport failed after source terminal');
+
+        expect(send).toHaveBeenCalledTimes(1);
+        expect(send).toHaveBeenCalledWith(
+            expect.objectContaining({
+                status: '500_SERVER_ERROR',
+                errorCode: 'AICODE_LIVE_SESSION_SOURCE_ERROR_V1',
+                response: 'streaming',
+            }),
+        );
+        expect(chunks).toEqual([]);
+        expect(patchLastAssistantMetaMock).toHaveBeenCalledTimes(1);
+        expect(appendChunkMock).toHaveBeenCalledTimes(1);
+    });
+
+    it('keeps a later meta.error as an ordinary body chunk after the 200 header', async
() => {
+

```

[AICode/src/extension/src/slimbk-local/endpoints/ai/orch/query/ask/ test /stream.endpoint.test.ts](#)

Copy

Review

Accept

  patching file:  [iaStreamHandlers.ts](#)

[AICode/src/extension/src/messageBus/handlers/ia/iaStreamHandlers.ts](#)

Copy

Review

Accept

PatchFull file

```
- import { isLiveSessionTerminalMetaChunk } from
'../../../../../sharedlib/types/liveSessionTypes/LiveSessionStream.util.js';
-
+ import { isLiveSessionTerminalMetaChunk } from
'../../../../../sharedlib/types/liveSessionTypes/LiveSessionStream.util.js';
+ import type { LiveSessionPostForwardingFinalizationResult } from
'../../../../../aiTools/orch/liveSession/registry/LiveSessionRecord.type.js';
+

-     let transportError: Error | undefined;
-     let errorCallbackHandled = false;
-     let decodedSourceError = false;
-
+     let transportError: Error | undefined;
+     let sourceErrorCallbackHandled = false;
+     let transportErrorCallbackHandled = false;
+

-         onError: (error: unknown): void => {
-             if (errorCallbackHandled) return;
-             errorCallbackHandled = true;
-
+         onError: (error: unknown): void => {
+
-             if (decoded) {
-                 decodedSourceError = true;
-
+             if (decoded) {
+                 if (sourceErrorCallbackHandled) return;
+                 sourceErrorCallbackHandled = true;
+
-             return;
-         }
-
+         return;
+     }
+
+     // A private V1 terminal and one later transport callback are
distinct events.
+     // Deduplicate each channel independently so terminal data stays
unique without
+     // hiding a network/iterator failure that happened after the
streaming header.
+     if (transportErrorCallbackHandled) return;
+     transportErrorCallbackHandled = true;
+

-     } catch (error) {
-         if (!decodedSourceError) {
-
+     } catch (error) {
+         // A thrown call after a V1 callback is also a real later transport failure.
Preserve the
+         // first transport diagnosis if the callback already supplied one before the
Promise rejected.
+         if (!transportError) {
+
-         await activityPublicationReady;
```

```

-     const finalization = liveSession.finalizeSourceAfterForwarding(runRef);
-     onAiStreamStop({ runRef, soundAttemptId });
-     await notifyPromptSourceSettled({
-         runRef,
-         runId: activity.runId,
-         finalization,
-     });
-
+     await activityPublicationReady;
+     let finalization:
+         | LiveSessionPostForwardingFinalizationResult
+         | undefined;
+     try {
+         // Registry finalization is the first terminal step because Activity and
cleanup consume its exact result.
+         finalization = liveSession.finalizeSourceAfterForwarding(runRef);
+     } catch (error) {
+         auxiliaryErrors.push(normalizeError(error));
+     }
+     try {
+         // Sound cleanup is independent from registry and coordinator failures; always
attempt it exactly once.
+         onAiStreamStop({ runRef, soundAttemptId });
+     } catch (error) {
+         auxiliaryErrors.push(normalizeError(error));
+     }
+     if (finalization) {
+         try {
+             // Coordinator settlement remains ordered after sound cleanup, but its
failure must not mask
+             // a transport, Activity, emit, or sound failure already collected for the
same source.
+             await notifyPromptSourceSettled({
+                 runRef,
+                 runId: activity.runId,
+                 finalization,
+             });
+         } catch (error) {
+             auxiliaryErrors.push(normalizeError(error));
+         }
+     }
+

```

[AI Code/src/extension/src/messageBus/handlers/ia/iaStreamHandlers.ts](#)

Copy

Review

Accept

  **patching file:**  [iaStreamHandlers.queryDoneSound.test.ts](#)

[AI Code/src/extension/src/messageBus/handlers/ia/ test /iaStreamHandlers.queryDoneSound.test.ts](#)

Copy

Review

Accept

PatchFull file

```

-     >();
-
+     >();
+ const soundStopMock = jest.fn<() => void>();
+ const notifyPromptSourceSettledMock = jest.fn<() => Promise<void>>();
+
-
-     onAiStreamChunk: () => order.push('soundChunk'),

```

```

-         onAiStreamStop: () => order.push('soundStop'),
-
+         onAiStreamChunk: () => order.push('soundChunk'),
+         onAiStreamStop: soundStopMock,
+
-     () => ({
-         notifyPromptSourceSettled: async () => order.push('coordinator'),
-
+     () => ({
+         notifyPromptSourceSettled: notifyPromptSourceSettledMock,
+
-     });
-
-     return { runId: 1, publication: Promise.resolve() };
-
+     return { runId: 1, publication: Promise.resolve() };
+     });
+     soundStopMock.mockReset();
+     soundStopMock.mockImplementation(() => {
+         order.push('soundStop');
+     });
+     notifyPromptSourceSettledMock.mockReset();
+     notifyPromptSourceSettledMock.mockImplementation(async () => {
+         order.push('coordinator');
+     });
+
-
-     it('decodes the first-header V1 source error exactly once and ignores a duplicate
onError callback', async () => {
-
+
+     it('decodes the first-header V1 source error exactly once and ignores a duplicate
V1 callback', async () => {
+
-         callbacks.onError(encoded);
-         callbacks.onError(new Error('duplicate callback'));
-
+         callbacks.onError(encoded);
+         callbacks.onError(encoded);
+
-
-     it('keeps the first generic transport error when onError is invoked repeatedly',
async () => {
-
+
+     it('keeps one V1 terminal when a later generic transport callback fails', async ()
=> {
+         registerIaStreamHandlers();
+         if (!handler) throw new Error('Expected registered stream handler.');
```

```

+     );
+     const emitted: unknown[] = [];
+
+     await expect(
+       handler(
+         {
+           turnId: 'turn',
+           sourceId: 'source',
+           threadId: 'a'.repeat(32),
+           storageScopeRelPath: '',
+           aiQuery: { type: 'prompt', promptValue: 'hello' },
+           aiQueryOptions: {},
+         },
+         (chunk) => emitted.push(chunk),
+         { htmlPage: 'prompt', webview },
+       ),
+     ).rejects.toThrow('network failed after V1 terminal');
+
+     expect(emitted).toHaveLength(1);
+     expect(finalizeSourceAfterForwardingMock).toHaveBeenCalledTimes(1);
+     expect(soundStopMock).toHaveBeenCalledTimes(1);
+     expect(notifyPromptSourceSettledMock).toHaveBeenCalledTimes(1);
+   });
+
+   it('keeps one V1 terminal and reports a later Slimbk body transport error', async
+   () => {
+     registerIaStreamHandlers();
+     if (!handler) throw new Error('Expected registered stream handler.');
```

```

+     const encoded = encodeLiveSessionSourceErrorV1({
+       reason: AiStopErrorReason.SYSTEM_ERROR,
+       message: 'system error',
+       details: 'provider stack',
+     });
+     callLocalApiMock.mockImplementationOnce(
+       async (_url, _args, callbacks): Promise<void> => {
+         callbacks.onError(encoded);
+         callbacks.onChunk({
+           status: '500_SERVER_ERROR',
+           errorCode: 'POST_V1_TRANSPORT_FAILURE',
+           errorMessage: 'transport failed after V1 terminal',
+         });
+       },
+     );
+     const emitted: unknown[] = [];
+
+     await expect(
+       handler(
+         {
+           turnId: 'turn',
+           sourceId: 'source',
+           threadId: 'a'.repeat(32),
+           storageScopeRelPath: '',
+           aiQuery: { type: 'prompt', promptValue: 'hello' },
+           aiQueryOptions: {},
+         },
+         (chunk) => emitted.push(chunk),
+         { htmlPage: 'prompt', webview },
+       ),
+     ).rejects.toThrow('transport failed after V1 terminal');
```

```

+
+     expect(emitted).toHaveLength(1);
+     expect(emitted[0]).toEqual(
+       expect.objectContaining({

```

```

+         type: 'meta',
+         value: expect.objectContaining({ metaType: 'error' })),
+     })),
+ );
+ expect(finalizeSourceAfterForwardingMock).toHaveBeenCalledTimes(1);
+ expect(soundStopMock).toHaveBeenCalledTimes(1);
+ expect(notifyPromptSourceSettledMock).toHaveBeenCalledTimes(1);
+ });
+
+ it('keeps the first generic transport error when onError is invoked repeatedly',
+ async () => {
+
+
+ });
+ });
+
+ it('aggregates transport, sound-stop, and coordinator failures without skipping
terminal steps', async () => {
+     registerIaStreamHandlers();
+     if (!handler) throw new Error('Expected registered stream handler.');
```

```

+     callLocalApiMock.mockImplementationOnce(
+         async (_url, _args, callbacks): Promise<void> => {
+             callbacks.onError(new Error('transport failed'));
+         },
+     );
+     soundStopMock.mockImplementationOnce(() => {
+         order.push('soundStop');
+         throw new Error('sound cleanup failed');
+     });
+     notifyPromptSourceSettledMock.mockImplementationOnce(async () => {
+         order.push('coordinator');
+         throw new Error('coordinator settlement failed');
+     });
+
+     let caught: unknown;
+     try {
+         await handler(
+             {
+                 turnId: 'turn',
+                 sourceId: 'source',
+                 threadId: 'a'.repeat(32),
+                 storageScopeRelPath: '',
+                 aiQuery: { type: 'prompt', promptValue: 'hello' },
+                 aiQueryOptions: {},
+             },
+             () => {},
+             { htmlPage: 'prompt', webview },
+         );
+     } catch (error) {
+         caught = error;
+     }
+
+     expect(caught).toBeInstanceOf(AggregateError);
+     if (!(caught instanceof AggregateError)) {
+         throw new Error('Expected ordered AggregateError.');
```

```

+     }
+     expect(
+         caught.errors.map((error: unknown) =>
+             error instanceof Error ? error.message : String(error),
+         ),
+     ).toEqual([
+         'transport failed',

```

```
+         'sound cleanup failed',
+         'coordinator settlement failed',
+     ]);
+     expect(order).toEqual(
+         expect.arrayContaining(['finalize', 'soundStop', 'coordinator']),
+     );
+     expect(finalizeSourceAfterForwardingMock).toHaveBeenCalledTimes(1);
+     expect(soundStopMock).toHaveBeenCalledTimes(1);
+     expect(notifyPromptSourceSettledMock).toHaveBeenCalledTimes(1);
+ });
+ });
+ 
```

[AI Code/src/extension/src/messageBus/handlers/ia/ test /iaStreamHandlers.queryDoneSound.test.ts](#)

Copy

Review

Accept

  patching file:  [liveSession.programming-doc-readme.md](#)

[AI Code/src/extension/src/aiTools/orch/liveSession/liveSession.programming-doc-readme.md](#)

Copy

Review

Accept

PatchFull file

Run epochs protect normal source replacement. A separate destructive generation protects Open against acquire/release claim ABA cycles. Durable Delete and EasyAgents Retry must acquire an opaque locator destruction claim, keep it through their destructive work, pass it to eviction, and release it in `finally`. Eviction never releases the claim.

Run epochs protect normal source replacement. A separate destructive generation protects Open against acquire/release claim ABA cycles. Durable Delete and EasyAgents Retry must acquire an opaque locator destruction claim, keep it through their destructive work, pass it to eviction, and release it in `finally`. Detached Prompt cleanup acquires the same locator-wide claim before its first asynchronous Abort/Await/ClearMemory step and keeps it through orphan cleanup, sound cleanup, trivial deletion, and Sidebar convergence. This mutual exclusion prevents a later Delete, Retry, Open, or source reservation from overtaking an already-started cleanup. The coordinator suppresses only the synchronous wake-up emitted by its own cleanup-claim release so a persistent cleanup failure cannot self-retry forever; external Delete/Retry releases still wake deferred cleanup. Eviction never releases the claim, and release retires both directions of the opaque claim lookup before emitting its wake-up.

Abort is claimed atomically for either an active observer lease or an exact source-owned reservation/record. Claiming synchronously latches `abortRequested`; the returned callback must be invoked without an intervening `await`. Reservation snapshots expose their real source/observer ownership so panel coordination and Activity rehydration never reinterpret an EasyAgents reservation as a detached normal Prompt source. The registry emits deeply immutable advisory wake-ups only after complete observer-inactive and destruction-claim-release mutations, and isolates listener failures. When the last observer lease disappeared before EasyAgents completed its business handoff, the handoff emits the observer-inactive wake-up again so deferred panel cleanup is retried after the final authority transition.

Abort is claimed atomically for either an active observer lease or an exact source-owned reservation/record. Claiming synchronously latches `abortRequested`; the returned callback must be invoked without an intervening `await`. A refused or failed frontend abort restores the exact still-active Stop capability and releases the local abort handshake so the operator may retry. When rendering already became idle but the transport capability remains stoppable, the handshake stays armed during the RPC and closes on its accepted response because no later `isStreaming` transition exists. A normal-Prompt

source Stop latch is also released when replay converges to a stable failure, because that surface must keep Retry usable after a pre-baseline source retirement. Reservation snapshots expose their real source/observer ownership so panel coordination and Activity rehydration never reinterpret an EasyAgents reservation as a detached normal Prompt source. The registry emits deeply immutable advisory wake-ups only after complete observer-inactive and destruction-claim-release mutations, and isolates listener failures. When the last observer lease disappeared before EasyAgents completed its business handoff, the handoff emits the observer-inactive wake-up again so deferred panel cleanup is retried after the final authority transition.

Only a first source ``meta.error`` may be encoded into the private V1 header protocol (``AICODE_LIVE_SESSION_SOURCE_ERROR_V1``). After a ``200_OK`` header, later ``meta.error`` values remain ordinary body chunks. Slimbk may serialize a post-header exception as a non-200 ``SlimbkResult`` in the body; handlers must detect that envelope before any ``AiStreamedChunk`` parsing and classify it as transport failure.

Only a first source ``meta.error`` may be encoded into the private V1 header protocol (``AICODE_LIVE_SESSION_SOURCE_ERROR_V1``). That non-200 header carries ``response='streaming'``: the generated local client reports the terminal immediately through ``onError`` but continues consuming the same body, and Slimbk keeps the writer open until endpoint settlement. A later source throw can therefore become a non-200 body envelope, preserving one visible terminal while still producing ``transportError``. After a ``200_OK`` header, later ``meta.error`` values remain ordinary body chunks. Slimbk may serialize a post-header exception as a non-200 ``SlimbkResult`` in the body; handlers must detect that envelope before any ``AiStreamedChunk`` parsing and classify it as transport failure. Final registry, sound, and panel-coordinator steps are attempted independently and join the same ordered failure aggregation instead of masking earlier transport or relay failures.

[AICode/src/extension/src/aiTools/orch/liveSession/liveSession.programming-doc-readme.md](#)

Copy

Review

Accept



patching file: [prompt-stream-dispatch-lock.programming-doc-readme.md](#)

[AICode/src/frontend/src/messageBus/prompt-stream-dispatch-lock.programming-doc-readme.md](#)

Copy

Review

Accept

PatchFull file

- Advances a reset generation so an older post-``await`` dispatch cannot remove or mutate successor work.
- Advances a reset generation so an older post-``await`` dispatch cannot remove or mutate successor work.
- Compares every incoming, queued, deferred, and withheld-suffix identity with the current synchronous attempt before any gating mutation. A late Queue A item from a reset attempt is resolved and discarded before it can lock ``PromptStreamLockStore`` for its successor.

The reset/finalization boundary is guarded by ``sourceId + streamInstanceId``. Each mounted transport listener freezes that identity in its callback object, so an old bus callback cannot relabel its frames with the current stream ref. A stale callback, continuation after ``await``, or Queue A ``runMerged`` item from an older accepted stream cannot reset Queue B or mutate the successor stream. ``completed`` and ``transportError`` are published only after Queue A idle, Queue B idle, and the final React commit barrier. If terminal data was already consumed before transport failure, no second error meta or suffix is injected.

The reset/finalization boundary is guarded by ``sourceId + streamInstanceId``. Each mounted transport listener freezes that identity in its callback object, so an old bus callback cannot relabel its frames with the current stream ref. A stale callback, continuation

after ``await``, or Queue A ``runMerged`` item from an older accepted stream cannot enter gating, acquire a lock, reset Queue B, or mutate the successor stream. ``completed`` and ``transportError`` are published only after Queue A idle, Queue B idle, and the final React commit barrier. If terminal data was already consumed before transport failure, no second error meta or suffix is injected.

[AICode/src/frontend/src/messageBus/prompt-stream-dispatch-lock.programming-doc-readme.md](#)

Copy

Review

Accept

  *patching file:*  [easyAgents.programming-doc-readme.md](#)

[AICode/src/extension/src/aiTools/easyAgents/easyAgents.programming-doc-readme.md](#)

Copy

Review

Accept

PatchFull file

- the targeted row must already own one durable child locator,
- the targeted child chat must not currently run one manual Prompt query,
- the targeted row must already own one durable child locator,
- the targeted child chat must not own an active liveSession source/reservation or one manual Prompt query,

-
3. rerun the canonical pending-line preparation ****before**** any destructive mutation,
 4. close any visible Prompt panel for the targeted child,
 5. destructively evict every retained generic liveSession run for that child locator,
 6. delete the old durable child chat,
 7. rebuild cold state from disk and confirm the line is back to ``pending``,
 8. re-check viewer presence,
 9. run the same strict Git cleanliness guard again on the current user branch outside ``.aicode/`` ****before**** switching back to the Batch branch,
 10. switch back to the persisted Batch branch and run the Git preflight,
 11. rerun the same canonical pending-line preparation ****again after the delete****,
 12. patch the parent to ``running``,
 13. create a retry live entry with ``activeRetryLineNumber``,
 14. relaunch the sequential loop for that one line only.
 3. rerun the canonical pending-line preparation ****before**** any destructive mutation,
 4. synchronously recheck registry and manual-Prompt authority immediately before claim acquisition,
 5. acquire the exact locator destruction claim with no intervening ``await``,
 6. close any visible Prompt panel for the targeted child,
 7. destructively evict every retained generic liveSession run for that child locator,
 8. delete the old durable child chat,
 9. rebuild cold state from disk and confirm the line is back to ``pending``,
 10. re-check viewer presence,
 11. run the same strict Git cleanliness guard again on the current user branch outside ``.aicode/`` ****before**** switching back to the Batch branch,
 12. switch back to the persisted Batch branch and run the Git preflight,
 13. rerun the same canonical pending-line preparation ****again after the delete****,
 14. patch the parent to ``running``,
 15. create a retry live entry with ``activeRetryLineNumber``,
 16. relaunch the sequential loop for that one line only.

-
- the first strict Git guard blocks Retry before any destructive mutation can delete the previous durable child,
 - the first strict Git guard blocks Retry before any destructive mutation can delete the previous durable child,
 - the child live-source/manual-query guard runs both before long preflight work and again in the same synchronous stack as claim acquisition; a source that appears during Git or preparation awaits therefore blocks Retry before panel close or eviction,
 - the exact destruction claim remains held through panel close, generic liveSession eviction, Activity removal, and durable child deletion, then releases in ``finally``,

[AICode/src/extension/src/aiTools/easyAgents/easyAgents.programming-doc-readme.md](#)

Copy
Review
Accept

  reading project:  e ["Ancien type Stop frontend"](#),  e ["Ancien API source Stop"](#),  e ["Ancien retour stopSource"](#),  e ["Mocks Stop générique"](#),  e ["Claim bool coordinateur"](#),  e ["Wrappers patch accidentels"](#),  [useAiStream.hook.ts](#) [L55-L155],  [useAiStream.hook.ts](#) [L680-L820],  [usePromptCancelController.hook.ts](#) [L1-L210],  [usePromptCancelController.hook.ts](#) [L300-L390],  [usePromptLiveSession.hook.ts](#) [L45-L180],  [usePromptLiveSession.hook.ts](#) [L350-L470],  [Prompt.tsx](#) [L650-L705],  [Prompt.autoRunPrompt.test.tsx](#) [L82-L100],  [PromptFallbackPersistBarrier.testHarness.tsx](#) [L198-L216],  [usePromptCancelController.stopCore.test.tsx](#) [L1-L330]

  patching file:  [usePromptCancelController.stopCore.test.tsx](#)
[AICode/src/frontend/src/app/prompt/Prompt/ test /usePromptCancelController.stopCore.test.tsx](#)
Grouped
([what is it?](#))

  reading project:  [usePromptRunController.code interpreter.status and deltas.test.ts](#) [L35-L65],  [usePromptRunController.mcp notice before gate.test.ts](#) [L75-L105]

  patching file:  [usePromptRunController.code interpreter.status and deltas.test.ts](#)
[AICode/src/frontend/src/app/prompt/Prompt/ test /usePromptRunController.code interpreter.status and deltas.test.ts](#)

Copy
Review
Accept
PatchFull file

```
-      lastToolCalls: lastToolCallsRef.current,  
-      stop: async () => {},  
-  
+      lastToolCalls: lastToolCallsRef.current,  
+      stop: async () => ({ accepted: true } ),  
+
```

[AICode/src/frontend/src/app/prompt/Prompt/ test /usePromptRunController.code interpreter.status and deltas.test.ts](#)

Copy
Review
Accept

  patching file:  [usePromptRunController.mcp notice before gate.test.ts](#)
[AICode/src/frontend/src/app/prompt/Prompt/ test /usePromptRunController.mcp notice before gate.test.ts](#)

Copy
Review
Accept
PatchFull file

```
-      lastToolCalls: lastToolCallsRef.current,  
-      stop: async () => {},  
-  
+      lastToolCalls: lastToolCallsRef.current,  
+      stop: async () => ({ accepted: true } ),  
+
```

[AICode/src/frontend/src/app/prompt/Prompt/ test /usePromptRunController.mcp notice before gate.test.ts](#)

Copy
Review
Accept

  reading project:  [PromptLiveSessionPanelCoordinator.service.ts](#) [L1-L320]

  patching file:  [PromptLiveSessionPanelCoordinator.service.ts](#)
[AICode/src/extension/src/extension/panels/prompt/liveSession/PromptLiveSessionPanelCoordinator.service.ts](#)
Grouped
([what is it?](#))

  reading project:  [PromptLiveSessionPanelCoordinator.service.test.ts](#) [L90-L500]
  patching file:  [PromptLiveSessionPanelCoordinator.service.test.ts](#)
[AICode/src/extension/src/extension/panels/prompt/liveSession/_test_/PromptLiveSessionPanelCoordinator.service.test.ts](#)
Grouped
([what is it?](#))

  reading project:  [stream.endpoint.ts](#) [L1-L230],  [iaStreamHandlers.ts](#) [L1-L300],  [usePromptStreamDispatchGateQueue.hook.ts](#) [L80-L180],  [usePromptStreamDispatchGateQueue.hook.ts](#) [L740-L1045],  [EasyAgentsBatchRunRetry.service.ts](#) [L210-L360],  [EasyAgentsBatchRunRetry.service.ts](#) [L540-L650],  [LiveSessionRegistry.service.ts](#) [L850-L885],  [PromptLiveSessionPanelCoordinator.service.test.ts](#) [L1-L100],  [easyAgents.programming-doc-readme.md](#) [L842-L905],  [liveSession.programming-doc-readme.md](#) [L15-L82],  [prompt-stream-dispatch-lock.programming-doc-readme.md](#) [L52-L140]
  scanning git:  ["Résumé des corrections"](#),  ["Contrôle whitespace"](#)
  scanning recent files
  reading project:  ["Types Stop compositeur"](#),  [usePromptRunController.hook.ts](#) [L1-L130]
  patching file:  [usePromptRunController.hook.ts](#)

[AICode/src/frontend/src/app/prompt/Prompt/usePromptRunController.hook.ts](#)

Copy

Review

Accept

PatchFull file

```
-      handleRetryQuery: () => AiStreamStartResult;  
-      handleStopFromComposer: () => Promise<void>;  
-  
+      handleRetryQuery: () => AiStreamStartResult;  
+      handleStopFromComposer: () => Promise<boolean>;  
+
```

[AICode/src/frontend/src/app/prompt/Prompt/usePromptRunController.hook.ts](#)

Copy

Review

Accept

  reading project:  ["Types API Stop source"](#),  [useAiStream.hook.promptSourceIdentity.test.ts](#) [L230-L390],  [iaStreamHandlers.queryDoneSound.test.ts](#) [L1-L170],  [usePromptLiveSession.normalPrompt.test.tsx](#) [L120-L255]

  patching file:  [usePromptLiveSession.normalPrompt.test.tsx](#)

[AICode/src/frontend/src/app/prompt/liveSession/core/_test_/usePromptLiveSession.normalPrompt.test.tsx](#)

Grouped

([what is it?](#))

  reading project:  [PromptStreamDispatchGateQueue.test.ts](#) [L1040-L1230],  ["Tests contrôleur annulation"](#),  [usePromptCancelController.confirmAfterMs.test.tsx](#) [L1-L320]

  patching file:  [usePromptCancelController.confirmAfterMs.test.tsx](#)

[AICode/src/frontend/src/app/prompt/Prompt/_test_/usePromptCancelController.confirmAfterMs.test.tsx](#)

Copy

Review

Accept

PatchFull file

```
- abortAckPending: boolean;
- handleStopCore: () => Promise<void>;
-
+ abortAckPending: boolean;
+ handleStopCore: () => Promise<boolean>;
+

- const stopCore = jest
-   .fn<() => Promise<void>>()
-   .mockResolvedValue(undefined);
-
+ const stopCore = jest
+   .fn<() => Promise<boolean>>()
+   .mockResolvedValue(true);
+

-   });
- });
-
+   });
+ });
+
+ test('usePromptCancelController: refused Stop keeps caller cleanup intact', async () =>
+ {
+   const host = document.createElement('div');
+   const root = createRoot(host);
+   const stopCore = jest
+     .fn<() => Promise<boolean>>()
+     .mockResolvedValue(false);
+   const afterStop = jest.fn<() => void>();
+   const apiRef: {
+     current: { stopFromComposer: () => Promise<void> } | undefined;
+   } = { current: undefined };
+
+   await act(async () => {
+     root.render(
+       <Harness
+         runningForEscape={true}
+         runningForConfirm={false}
+         abortAckPending={false}
+         handleStopCore={stopCore}
+         onAfterCancelCleanup={afterStop}
+         apiRef={apiRef}
+       />,
+     );
+   });
+   if (!apiRef.current) {
+     throw new Error('Expected cancel controller API to be available');
+   }
+
+   await act(async () => {
+     await apiRef.current?.stopFromComposer();
+   });
+
+   expect(stopCore).toHaveBeenCalledTimes(1);
+   expect(afterStop).not.toHaveBeenCalled();
+
+   await act(async () => {
+     root.unmount();
+   });
+ });
```

+
[AICode/src/frontend/src/app/prompt/Prompt/ test /usePromptCancelController.confirmAfterMs.test.tsx](#)
Copy
Review
Accept

  reading project:  e ["Types callback Stop core"](#),  e ["Appels directs Stop stream"](#),  [AiStreamStartArgs.type.ts](#) [L1-L100],  e ["Test erreur Slimbk body"](#),  [iaStreamHandlers.queryDoneSound.test.ts](#) [L285-L375],  e ["Type contrôleur Prompt tests"](#),  [aiStopSuffix.util.ts](#) [L70-L190],  [aiStopSuffix.util.ts](#) [L190-L285],  e ["Générateur client stream"](#),  g ["Docs compilateur Slimbk"](#),  [slimbkBuildHeader.ts](#) [L320-L430],  [slimbkBuildHeader.errorCode.test.ts](#),  e ["Slimbk result tests"](#),  e ["Branches header générées"](#),  [slimbkBuildHeader.ts](#) [L200-L270],  [slimbkBuildHeader.ts](#) [L130-L210],  [parseSlimbkStreamBodyError.util.ts](#),  [slimbkBuildHeader.ts](#) [L1-L100],  g ["Tests framework Slimbk"](#),  [aiStopSuffix.util.ts](#) [L40-L85],  [aiStopSuffix.util.test.ts](#) [L80-L154],  l ["Tests runtime Slimbk"](#),  [parseSlimbkStreamBodyError.util.test.ts](#),  [Slimbk.ts](#) [L1-L90],  [SlimbkEndpoint.ts](#) [L1-L120],  [SlimbkEndpoint.ts](#) [L120-L240]
  patching file:  [SlimbkResult.type.ts](#)

[AICode/src/sharedlib/src/slimbk/framework/SlimbkResult.type.ts](#)

Copy
Review
Accept

PatchFull file

```
- *      - For normal response: this type exactly
- *      - For stream response: The first line is this type with response='stream'
-
+ *      - For normal response: this type exactly
+ *      - For stream response: The first line is this type with response='streaming'
+
```

```
-      errorCode?: string;
-
+      errorCode?: string;
+      /**
+       * Keep a streamed response open after reporting this header error to the
client.
+       *
+       * This is reserved for protocols whose first application item is represented
by the
+       * header while later body transport failures must remain observable on the
same stream.
+       */
+      response?: 'streaming';
+
```

[AICode/src/sharedlib/src/slimbk/framework/SlimbkResult.type.ts](#)

Copy
Review
Accept

  patching file:  [Slimbk.ts](#)

[AICode/src/sharedlib/src/slimbk/framework/Slimbk.ts](#)

Copy
Review
Accept

PatchFull file

```
-      */
-      function sendSlimbkResult(result: SlimbkResult): void {
-
-
```

```

+      */
+      function sendSlimbkResult(
+          result: SlimbkResult,
+          allowStreamingErrorHandler = false,
+      ): void {
+
+
+      // On error result: end the stream and optionally throw
+      if (result.status !== '200_OK') {
+
+      // Ordinary errors terminate immediately. A streaming error header reports
+      one // application terminal through the header while deliberately retaining the
+      body // channel for any later transport envelope produced by the same endpoint
+      execution.
+      if (
+          result.status !== '200_OK' &&
+          !(
+              allowStreamingErrorHandler &&
+              result.response === 'streaming'
+          )
+      ) {
+
+      // First chunk = full SlimbkResult
+      sendSlimbkResult(result);
+
+      // First chunk = full SlimbkResult
+      sendSlimbkResult(result, true);
+

```

[AI Code/src/sharedlib/src/slimbk/framework/Slimbk.ts](#)

Copy

Review

Accept

  patching file:  [aiStopSuffix.util.ts](#)

[AI Code/src/sharedlib/src/types/aiTypes/aiStopSuffix.util.ts](#)

Copy

Review

Accept

PatchFull file

```

-      readonly errorCode: typeof LIVE_SESSION_SOURCE_ERROR_V1_CODE;
-
+      readonly errorCode: typeof LIVE_SESSION_SOURCE_ERROR_V1_CODE;
+      readonly response: 'streaming';
+
+
+      errorCode: LIVE_SESSION_SOURCE_ERROR_V1_CODE,
+
+      errorCode: LIVE_SESSION_SOURCE_ERROR_V1_CODE,
+      response: 'streaming',
+

```

[AI Code/src/sharedlib/src/types/aiTypes/aiStopSuffix.util.ts](#)

Copy

Review

Accept

  patching file:  [aiStopSuffix.util.test.ts](#)

[AICode/src/sharedlib/src/types/aiTypes/ test /aiStopSuffix.util.test.ts](#)

Copy

Review

Accept

PatchFull file

```
-         expect(encoded.errorCode).toBe(LIVE_SESSION_SOURCE_ERROR_V1_CODE);
-
+         expect(encoded.errorCode).toBe(LIVE_SESSION_SOURCE_ERROR_V1_CODE);
+         expect(encoded.response).toBe('streaming');
+
```

[AICode/src/sharedlib/src/types/aiTypes/ test /aiStopSuffix.util.test.ts](#)

Copy

Review

Accept



patching file: [slimbkBuildHeader.ts](#)

[AICode/src/sharedlib/src/slimbk/compiler/generateClient/slimbkBuildHeader.ts](#)

Copy

Review

Accept

PatchFull file

```
- *      - For normal response: this type exactly
- *      - For stream response: The first line is this type with response='stream'
-
+ *      - For normal response: this type exactly
+ *      - For stream response: The first line is this type with response='streaming'
+
```

```
-         errorCode?: string;
-
+         errorCode?: string;
+         /** Preserve the body channel after this non-200 stream header. */
+         response?: 'streaming';
+
```

```
-     }
-     // Evaluate status *outside* of the JSON.parse try/catch; propagate remote
errors cleanly.
-
+     }
+     // Evaluate status *outside* of the JSON.parse try/catch; report remote errors
cleanly.
+
```

```
-         if (header.status !== '200_OK') {
-             throw new SlimbkCallError({
-
+         if (header.status !== '200_OK') {
+             const remoteError = new SlimbkCallError({
+
+
```

```
-             errorCode: header.errorCode,
-             });
-
+             errorCode: header.errorCode,
+             });
+             callbacks.onError(remoteError);
+             if (header.response !== 'streaming') {
+                 return;
+             }
+
```

[AICode/src/sharedlib/src/slimbk/compiler/generateClient/slimbkBuildHeader.ts](#)

Copy
Review
Accept

  patching file:  [slimbkBuildHeader.errorCode.test.ts](#)

[AICode/src/sharedlib/src/slimbk/compiler/generateClient/ test /slimbkBuildHeader.errorCode.test.ts](#)

Error

([what is it?](#))

  patching file:  [slimbkBuildHeader.errorCode.test.ts](#)

[AICode/src/sharedlib/src/slimbk/compiler/generateClient/ test /slimbkBuildHeader.errorCode.test.ts](#)

Copy

Review

Accept

PatchFull file

```
- * - Ensure the generated header transports a structured `errorCode` field end-to-end.  
- * - Ensure the generated client throws a structured `SlimbkCallError` instead of  
-
```

```
+ * - Ensure the generated header transports a structured `errorCode` field end-to-end.  
+ * - Ensure the generated client propagates a structured `SlimbkCallError` instead of  
+
```

```
- const text = slimbkBuildHeader(buildParams('cloud'));  
-  
- // Transport: SlimbkResult error variant includes a dedicated errorCode field.  
- expect(text).toContain('errorCode?: string;');
```

```
+ const text = slimbkBuildHeader(buildParams('cloud'));  
+  
+ // Transport: SlimbkResult error variant includes a dedicated errorCode field.  
+ expect(text).toContain('errorCode?: string;');  
+ expect(text).toContain("response?: 'streaming'");  
+
```

```
-  
- it('includes errorCode in SlimbkResult and throws SlimbkCallError (local)', () => {  
-  
+  
+ it('includes errorCode in SlimbkResult and reports SlimbkCallError (local)', () =>  
+ {  
+  
+
```

```
- expect(text).toContain('errorCode?: string;');  
-  
- // Structured error type exists.  
-  
+ expect(text).toContain('errorCode?: string;');  
+ expect(text).toContain("response?: 'streaming'");  
+  
+ // Structured error type exists.  
+
```

```
-  
- // Local callSlimbkStream propagates remote header errors via SlimbkCallError  
too.  
-
```

```
+  
+ // Local callSlimbkStream reports the structured error and keeps explicitly  
streaming headers open.  
+
```

```
- expect(text).toContain(  
-
```

```

-         "if (header.status !== '200_OK') {\n                throw new
SlimbkCallError({"
-
+         expect(text).toContain(
+         "if (header.status !== '200_OK') {\n                const remoteError = new
SlimbkCallError({"
+
-         "if (header.status !== '200_OK') {\n                const remoteError = new
SlimbkCallError({"
-         );
-
+         "if (header.status !== '200_OK') {\n                const remoteError = new
SlimbkCallError({"
+         );
+         expect(text).toContain('callbacks.onError(remoteError);');
+         expect(text).toContain("if (header.response !== 'streaming') {");
+

```

[AICode/src/sharedlib/src/slimbk/compiler/generateClient/_test_/slimbkBuildHeader.errorCode.test.ts](#)

Copy

Review

Accept

  **patching file:**  [stream.endpoint.ts](#)

[AICode/src/extension/src/slimbk-local/endpoints/ai/orch/query/ask/stream.endpoint.ts](#)

Grouped

([what is it?](#))

  **patching file:**  [stream.endpoint.test.ts](#)

[AICode/src/extension/src/slimbk-local/endpoints/ai/orch/query/ask/_test_/stream.endpoint.test.ts](#)

Grouped

([what is it?](#))

  **writing file:**  [Slimbk.streamingErrorHandler.test.ts](#)

[AICode/src/sharedlib/src/slimbk/framework/_test_/Slimbk.streamingErrorHandler.test.ts](#)

Copy

Review

Accept

```

import { describe, expect, it, jest } from '@jest/globals';
import z from 'zod';

```

```

import { Slimbk } from '../Slimbk.js';
import { SlimbkEndpoint } from '../SlimbkEndpoint.js';

```

```

describe('Slimbk streaming error headers', () => {
  it('keeps an explicitly streaming non-200 header open for later body transport
errors', async () => {
    const consoleLogSpy = jest
      .spyOn(console, 'log')
      .mockImplementation(() => {});
    const consoleErrorSpy = jest
      .spyOn(console, 'error')
      .mockImplementation(() => {});
    try {
      const slimbk = new Slimbk();
      const endpoint = new SlimbkEndpoint({
        respMode: 'stream',
        argsSchema: z.object({}),

```

header, but the
fully settles.

```
handler: async (_params, sendStreamResult) => {
  // The first application terminal is represented by the error

  // returned writer must remain usable until the endpoint execution

  const writeBody = sendStreamResult({
    status: '500_SERVER_ERROR',
    errorCode: 'STREAMING_TERMINAL',
    response: 'streaming',
    errorMessage: 'terminal application error',
  });
  writeBody(
    JSON.stringify({
      turnId: 'turn-1',
      type: 'token',
      value: 'later body data',
    }),
  );
  throw new Error('later transport failure');
},
}).setRoute('/api/test/streaming-error-header');
slimbk.addEndpoint(endpoint);

const stream = await slimbk.call({
  route: '/api/test/streaming-error-header',
  headers: {},
  args: {},
});
let text = '';
for await (const chunk of stream) {
  text += Buffer.isBuffer(chunk)
    ? chunk.toString('utf8')
    : String(chunk);
}
const envelopes = text
  .trim()
  .split('\n')
  .map((line): unknown => JSON.parse(line));

expect(envelopes).toHaveLength(3);
expect(envelopes[0]).toEqual({
  status: '500_SERVER_ERROR',
  errorCode: 'STREAMING_TERMINAL',
  response: 'streaming',
  errorMessage: 'terminal application error',
});
expect(envelopes[1]).toEqual({
  turnId: 'turn-1',
  type: 'token',
  value: 'later body data',
});
expect(envelopes[2]).toEqual(
  expect.objectContaining({
    status: '500_SERVER_ERROR',
    errorMessage: expect.stringContaining(
      'later transport failure',
    ),
  }),
);
} finally {
  consoleLogSpy.mockRestore();
}
```

```
        consoleErrorSpy.mockRestore();
    }
});
});
```

[AICode/src/sharedlib/src/slimbk/framework/_test_/Slimbk.streamingErrorHandler.test.ts](#)

Copy

Review

Accept

  reading project:  e "[Règle require await](#)",  [slimbkBuildHeader.ts](#) [L40-L90],  [slimbkBuildHeader.ts](#) [L380-L430]

  patching file:  [liveSession.programming-doc-readme.md](#)

[AICode/src/extension/src/aiTools/orch/liveSession/liveSession.programming-doc-readme.md](#)

Grouped

([what is it?](#))

  patching file:  [slimbkBuildHeader.errorCode.test.ts](#)

[AICode/src/sharedlib/src/slimbk/compiler/generateClient/_test_/slimbkBuildHeader.errorCode.test.ts](#)

Grouped

([what is it?](#))

  reading project:  [eslint.config.ts](#) [L35-L62],  [SlimbkResult.type.ts](#) [L1-L80],  [Slimbk.ts](#) [L90-L135],  [slimbkBuildHeader.errorCode.test.ts](#) [L1-L100],  [Slimbk.streamingErrorHandler.test.ts](#),  [stream.endpoint.test.ts](#) [L230-L390],  [EasyAgentsBatchRunRetry.guards.service.test.ts](#) [L80-L230],  [iaStreamHandlers.queryDoneSound.test.ts](#) [L550-L650],  [usePromptCancelController.confirmAfterMs.test.tsx](#) [L35-L280],  [usePromptLiveSession.normalPrompt.test.tsx](#) [L125-L260],  e "[Attentes enveloppe V1](#)",  [iaStreamHandlers.queryDoneSound.test.ts](#) [L390-L470],  [iaStreamHandlers.queryDoneSound.test.ts](#) [L470-L535]

  patching file:  [iaStreamHandlers.queryDoneSound.test.ts](#)

[AICode/src/extension/src/messageBus/handlers/ia/_test_/iaStreamHandlers.queryDoneSound.test.ts](#)

Grouped

([what is it?](#))

  reading project:  [slimbkBuildHeader.ts](#) [L228-L260],  e "[Snapshots client généré](#)",  r "[handleStopFromComposer](#)"

  patching file:  [iaStreamHandlers.ts](#)

[AICode/src/extension/src/messageBus/handlers/ia/iaStreamHandlers.ts](#)

Grouped

([what is it?](#))

  patching file:  [iaStreamHandlers.queryDoneSound.test.ts](#)

[AICode/src/extension/src/messageBus/handlers/ia/_test_/iaStreamHandlers.queryDoneSound.test.ts](#)

Grouped

([what is it?](#))

  reading project:  e "[Doc isomorphisme V1](#)",  h "[Doc erreurs live Prompt](#)",  [isomorphism-invariant-rule-between-streaming-and-rehydration-of-chat-sessions.programming-doc-readme.md](#) [L1-L45]

  patching file:  [isomorphism-invariant-rule-between-streaming-and-rehydration-of-chat-sessions.programming-doc-readme.md](#)
[AICode/src/frontend/src/app/prompt/hydration/isomorphism-invariant-rule-between-streaming-and-rehydration-of-chat-sessions.programming-doc-readme.md](#)

Copy

Review

Accept

PatchFull file

- `replaceTurnAssistants` removes only assistants from the exact logical turn;
`continueAssistant` overlays the immutable source-start message baseline.
- The first source `meta.error` may arrive through the private V1 header protocol. Later
`meta.error` chunks remain ordinary body data.
- `replaceTurnAssistants` removes only assistants from the exact logical turn;
`continueAssistant` overlays the immutable source-start message baseline.
- The first source `meta.error` may arrive immediately through the private V1 non-200
header with `response='streaming'`; the client publishes that terminal once while
continuing the same body so a later transport envelope can still produce
`outcome='transportError'`. Later `meta.error` chunks remain ordinary body data.
[AICode/src/frontend/src/app/prompt/hydration/isomorphism-invariant-rule-between-streaming-and-rehydration-of-chat-sessions.programming-doc-readme.md](#)

Copy

Review

Accept

  patching file:  [SlimbkResult.type.ts](#)
[AICode/src/sharedlib/src/slimbk/framework/SlimbkResult.type.ts](#)

Grouped

([what is it?](#))

  patching file:  [slimbkBuildHeader.ts](#)
[AICode/src/sharedlib/src/slimbk/compiler/generateClient/slimbkBuildHeader.ts](#)

Grouped

([what is it?](#))

  reading project:  [usePromptRunController.hook.ts](#) [L85-L105],  [usePromptCancelController.hook.ts](#) [L215-L270],  e "[Usages ATTEMPT B Queue](#)"

  patching file:  [useAiStream.hook.ts](#)
[AICode/src/frontend/src/messageBus/useAiStream.hook.ts](#)

Grouped

([what is it?](#))

  patching file:  [usePromptCancelController.hook.ts](#)
[AICode/src/frontend/src/app/prompt/Prompt/usePromptCancelController.hook.ts](#)

Grouped

([what is it?](#))

  patching file:  [iaStreamHandlers.queryDoneSound.test.ts](#)
[AICode/src/extension/src/messageBus/handlers/ia/ test /iaStreamHandlers.queryDoneSound.test.ts](#)

Grouped

([what is it?](#))

  reading project:  e "[Mock service openPanel](#)",  [openPanel.util.test.ts](#) [L1-L150]

  patching file:  [easyAgents.programming-doc-readme.md](#)
[AICode/src/extension/src/aiTools/easyAgents/easyAgents.programming-doc-readme.md](#)
Grouped
([what is it?](#))

  patching file:  [liveSession.programming-doc-readme.md](#)
[AICode/src/extension/src/aiTools/orch/liveSession/liveSession.programming-doc-readme.md](#)
Grouped
([what is it?](#))

  reading project:  [PromptLiveSessionPanelCoordinator.service.ts](#) [L20-L320]
  patching file:  [usePromptRunController.code interpreter.status and deltas.test.ts](#)
[AICode/src/frontend/src/app/prompt/Prompt/_test_/usePromptRunController.code interpreter.status and deltas.test.ts](#)
Grouped
([what is it?](#))

  patching file:  [usePromptRunController.mcp notice before gate.test.ts](#)
[AICode/src/frontend/src/app/prompt/Prompt/_test_/usePromptRunController.mcp notice before gate.test.ts](#)
Grouped
([what is it?](#))

  reading project:  e ["Mocks typés contrôleur run"](#)
  patching file:  [usePromptLiveSession.hook.ts](#)
[AICode/src/frontend/src/app/prompt/liveSession/core/usePromptLiveSession.hook.ts](#)
Error
([what is it?](#))

  reading project:  [usePromptLiveSession.hook.ts](#) [L270-L385]
  patching file:  [usePromptLiveSession.hook.ts](#)
[AICode/src/frontend/src/app/prompt/liveSession/core/usePromptLiveSession.hook.ts](#)
Grouped
([what is it?](#))

  reading project:  [Slimbk.ts](#) [L95-L125],  [Slimbk.ts](#) [L255-L285]
  patching file:  [Slimbk.ts](#)
[AICode/src/sharedlib/src/slimbk/framework/Slimbk.ts](#)
Grouped
([what is it?](#))

  scanning recent files
  patching file:  [usePromptCancelController.hook.ts](#)
[AICode/src/frontend/src/app/prompt/Prompt/usePromptCancelController.hook.ts](#)
Grouped
([what is it?](#))

  patching file:  [usePromptCancelController.stopCore.test.tsx](#)
[AICode/src/frontend/src/app/prompt/Prompt/_test_/usePromptCancelController.stopCore.test.tsx](#)
Grouped

([what is it?](#))



patching file: [liveSession.programming-doc-readme.md](#)

[AICode/src/extension/src/aiTools/orch/liveSession/liveSession.programming-doc-readme.md](#)

Grouped

([what is it?](#))



reading project: [usePromptCancelController.hook.ts](#) [L105-L225]



patching file: [usePromptCancelController.hook.ts](#)

[AICode/src/frontend/src/app/prompt/Prompt/usePromptCancelController.hook.ts](#)

Grouped

([what is it?](#))



patching file: [usePromptCancelController.hook.ts](#)

[AICode/src/frontend/src/app/prompt/Prompt/usePromptCancelController.hook.ts](#)

Grouped

([what is it?](#))



patching file: [usePromptCancelController.hook.ts](#)

[AICode/src/frontend/src/app/prompt/Prompt/usePromptCancelController.hook.ts](#)

Grouped

([what is it?](#))

Corrections complètes appliquées

Joel, **tous les correctifs identifiés pendant la vérification ont été implémentés**. Les courses avaient beaucoup de pistes pour s'échapper ; elles viennent de perdre leur permis de conduire. 🚗🔒

L'intervention couvre **36 fichiers source, tests et programming docs**, avec **un nouveau test runtime Slimbk**. Aucun fichier *.generated.* n'a été modifié manuellement : seul le générateur source concerné a été corrigé.

Résultat par lot

Lot	Corrections réalisées	Couverture de régression ajoutée	Complétude
Stop et Retry frontend	Récupération après refus/échec RPC, handshake transactionnel, déblocage du Retry après retraite pré-baseline	Stop refusé, erreur transport, double clic, Stop post-render, Stop pré-baseline puis Retry	
Cleanup et claims destructifs	Cleanup détaché protégé par un claim exact, suppression du TOCTOU, prévention des boucles de wake-up	Claim tenu pendant les opérations asynchrones, release propre, wake-up propre/externe	
Retry EasyAgents	Seconde vérification synchrone de l'autorité juste avant l'acquisition du claim	Source apparue pendant les préchecks asynchrones	
Queue B	Rejet des entrées stale avant toute mutation de lock, y compris suffixes et buffers différés	Ancien item Queue A arrivant après reset et tentative successeur	
Transport V1/Slimbk	Header V1 immédiatement visible mais stream maintenu ouvert pour une erreur transport ultérieure	Header non-200 streaming, terminal V1 + body error, terminal V1 + callback transport	
Finalisation extension	Agrégation déterministe transport, Activity, son et coordinateur	Échecs combinés sans étape terminale omise	
Documentation	Invariants Stop, claims, Retry,	Alignement	

Lot	Corrections réalisées	Couverture de régression ajoutée	Complétude
	Queue B, isomorphisme et transport actualisés	maintenance/code	

Stop, Retry et handshake frontend 🚫

Le contrat de Stop dispose maintenant d'un résultat strict `{ accepted: true | false }` dans `AiStreamStartArgs.type.ts`.

Les changements principaux sont les suivants :

- `useAiStream.hook.ts` :
 - consomme toujours la capacité Stop avant le premier `await` ;
 - restaure cette capacité si le RPC est refusé ou échoue ;
 - ne la restaure que si la même tentative exacte possède encore le stream ;
 - retourne un résultat explicite au contrôleur ;
 - préserve le teardown terminal normal.
- `usePromptCancelController.hook.ts` :
 - distingue maintenant une annulation réellement acceptée d'un refus ;
 - ne supprime plus les notices ou l'état utilisateur après un Stop refusé ;
 - protège le handshake pendant le RPC ;
 - termine correctement le handshake d'un Stop post-render dès la réponse acceptée ;
 - laisse un terminal actif fermer le handshake même si le RPC est encore en vol.
- `usePromptLiveSession.hook.ts` :
 - libère le latch Stop lorsqu'un Open converge vers `liveSessionReplayFailed` ;
 - libère également le latch sur handoff direct, transport error ou completion sans terminal ;
 - rend immédiatement le Retry disponible après une retraite pré-baseline ;
 - retourne le résultat réel du Stop source.
- `Prompt.tsx` :
 - utilise désormais un propriétaire unique pour le Stop normalPrompt ;
 - route toutes les phases normalPrompt, y compris l'observer reconnecté, vers le Stop source exact ;
 - conserve le contrôleur générique pour Prompt classique et les observers EasyAgents.

Cleanup et claims destructifs 🗑️

Le cleanup détaché dans `PromptLiveSessionPanelCoordinator.service.ts` acquiert maintenant son propre claim destructif exact **avant le premier `await`**.

Ce claim reste détenu pendant :

- `Abort` → `AwaitStop` → `ClearMemory` ;
- le nettoyage des attachments orphelins ;
- le cleanup du son ;
- le cleanup trivial du chat ;
- la convergence du SidebarStore.

Cela empêche désormais un Delete, un Retry, un Open ou une nouvelle source de dépasser un cleanup déjà commencé.

Le coordinateur distingue également :

- les releases externes, qui réveillent réellement le cleanup différé ;
- la release de son propre claim, dont le wake-up synchrone est supprimé afin d'éviter une boucle infinie si un cleanup échoue de façon persistante.

Dans `LiveSessionRegistry.service.ts`, la release d'un claim supprime maintenant aussi son reverse lookup opaque. Le registre ne conserve donc plus une entrée morte après chaque claim libéré.

Retry EasyAgents atomique

Dans `EasyAgentsBatchRunRetry.service.ts` :

1. l'autorité registre/manual Prompt est contrôlée avant les préchecks longs ;
2. les vérifications Git et la préparation canonique peuvent s'exécuter ;
3. l'autorité est **recontrôlée synchroniquement** ;
4. le claim est acquis immédiatement, sans aucun `await` intermédiaire ;
5. le claim bloque alors toute nouvelle source jusqu'à la fin du Retry.

Une source apparue pendant les opérations Git ou de préparation ne peut donc plus être évincée accidentellement par un précheck devenu périmé.

Queue B et protection du successeur

[usePromptStreamDispatchGateQueue.hook.ts](#) compare maintenant l'identité { `sourceId`, `streamInstanceId` } avec l'autorité active :

- dès l'entrée provenant de Queue A ;
- avant le traitement d'une tête déjà en queue ;
- avant le replay d'un chunk différé ;
- avant le replay d'un suffixe `tool_calls`.

Une entrée stale est résolue silencieusement afin de libérer la backpressure, mais elle est supprimée **avant** de pouvoir :

- acquérir `PromptStreamLockStore` ;
- modifier `seenCallIds` ;
- publier un préfixe `tool_calls` ;
- contaminer les buffers du successeur.

Protocole V1 et erreur transport ultérieure

La correction ne retarde plus le premier terminal `meta.error`.

Le contrat `Slimbk` expose maintenant un header d'erreur streamable dans :

- [SlimbkResult.type.ts](#)
- [Slimbk.ts](#)
- [slimbkBuildHeader.ts](#)

Le déroulement est désormais :

1. le premier `meta.error` est immédiatement envoyé via le header V1 non-200 ;
2. ce header contient `response='streaming'` ;
3. le client local publie immédiatement l'erreur via `onError` ;
4. il continue néanmoins à lire le body ;
5. si la source lève ensuite une erreur, `Slimbk` l'écrit sous forme d'enveloppe non-200 dans le body ;
6. le handler conserve un seul terminal visible et publie `transportError`.

L'encodeur V1 de [aiStopSuffix.util.ts](#) produit maintenant ce contrat streamable.

L'endpoint [stream.endpoint.ts](#) conserve son émission immédiate du premier terminal, tout en gardant le writer disponible pour une éventuelle enveloppe transport ultérieure.

Le générateur a été corrigé à la source ; **aucun client généré n'a été édité manuellement.**

Finalisation et agrégation des erreurs

Dans [iaStreamHandlers.ts](#) :

- les callbacks V1 et les callbacks transport disposent maintenant de gardes d'idempotence distinctes ;
- un terminal V1 n'empêche plus une erreur réseau ultérieure d'être enregistrée ;
- la finalisation registre, l'arrêt du son et la notification du coordinateur sont tous tentés ;
- leurs erreurs rejoignent l'agrégation existante sans masquer l'erreur transport initiale ;
- l'ordre déterministe est conservé : transport, erreurs auxiliaires, son, coordinateur.

Couverture de régression ajoutée

Les nouveaux scénarios couvrent notamment :

- Stop accepté avant baseline, source retirée, Open `not_found`, puis Retry réussi ;
- refus et exception du RPC Stop avec restauration de la capacité ;
- Stop post-render dont le handshake se termine sur la réponse RPC ;
- entrée Queue A stale reçue après le reset du successeur ;
- source `EasyAgents` apparue pendant les préchecks asynchrones ;
- claim détenu pendant tout le cleanup détaché ;
- suppression du wake-up récursif lors de la release du claim de cleanup ;
- terminal V1 suivi d'un callback transport ;
- terminal V1 suivi d'une enveloppe `Slimbk` body ;
- combinaison transport + erreur du son + erreur du coordinateur ;
- fonctionnement runtime d'un header `Slimbk` non-200 gardant son body ouvert.

Un test runtime dédié a été créé dans [Slimbk.streamingErrorHandler.test.ts](#).

Les principaux tests enrichis sont :

- [usePromptLiveSession.normalPrompt.test.tsx](#)
- [useAiStream.hook.promptSourceIdentity.test.ts](#)
- [usePromptCancelController.stopCore.test.tsx](#)
- [PromptStreamDispatchGateQueue.test.ts](#)

- Documentation de maintenance actualisée** 
- Les nouveaux invariants sont documentés dans :

Les nouveaux invariants sont documentés dans :

- ### Nom du commit

Copy

🎵 Les courses sont maintenant verrouillées 🛑 Les Stops peuvent respirer, et le replay continuer sans dérailler 🎵

<USER>

</USER>

  *scanning problems*

L60], e "[*Ancienne API Stop test*](#)", e "[*Promesses Stop obsolètes*](#)", e "[*Ancien retour Stop hook*](#)"

[AICode/src/frontend/src/messageBus/ test /useAiStream.hook.liveSessionObserver.test.ts](#)

Review

Accept

PatchFull file

```
+   if (!activeApi) throw new Error('Active hook API was not published.');
```

[AICode/src/frontend/src/messageBus/ test /useAiStream.hook.liveSessionObserver.test.ts](#)

Review

Accept

  reading project:  [useAiStream.hook.liveSessionObserver.test.ts](#) [L1-L25],  [useAiStream.hook.liveSessionObserver.test.ts](#) [L190-L215],  [useAiStream.hook.liveSessionObserver.test.ts](#) [L215-L225]

Erreur TypeScript corrigée

Tu as bien fait de la signaler, Joel. Le test utilisait encore l'ancien contrat de `stop()` :

- **ancien contrat** : `Promise<void>`
- **nouveau contrat strict** : `Promise<AiStreamStopResult>`

J'ai corrigé [useAiStream.hook.liveSessionObserver.test.ts](#) en :

1. important le type `AiStreamStopResult` ;
2. remplaçant `Promise<void>` par `Promise<AiStreamStopResult>` ;
3. vérifiant explicitement que le premier `Stop` retourne `{ accepted: true }`.

Correction

État

Import du nouveau type `Stop`



Type de `firstStop` synchronisé avec la production



Assertion du résultat accepté



Second `Stop` same-stack toujours protégé



Nom du commit

text

Copy

`fix(live-session): align observer stop test with strict result contract`

 Le type du `Stop` est recalé  Le test et le hook peuvent de nouveau compiler accordés 

</ASSISTANT>

SHOW ALLPage 1 / 1