Episode Book

EPISODE #5814

Who Remembers Where You Left Off?

PUBLISHED	9 October 2026 at 14:29 UTC
DURATION	21:46
LISTEN	myweirdprompts.com/episode/playback-resume-state-libraries/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.2
SOURCES	12 cited by the research (listed at the end)
BOOK GENERATED	19 October 2026 at 17:40 UTC

SYNOPSIS

Daniel asked whether playback progress, completion status, and listening history are reusable components or something every media app rebuilds from scratch. The answer sits somewhere uncomfortable: mpv bundles a real resume feature (quit-watch-later, save-position-on-quit, flat watch-later config files), but it's locked inside the player. Media3 and ExoPlayer hand you an event stream, a position, and a maintainer telling you to persist it yourself — in Room, Android's SQLite layer, which knows nothing about audio. GstPlay exposes position, duration, and a message bus, but no resume store, and its Play API is documented as unstable. MPRIS standardizes live playback state beautifully over D-Bus — position, status, seek, track list — but explicitly has no concept of a resume position across sessions.

The pattern: standards exist for control, not for memory. There is no portable schema for playback history, no shared vocabulary that would let one player ask another where you were. Specialized playback-history libraries don't appear to exist as standalone components — only app-level implementations and thin wrappers.

TRANSCRIPT



Corn

Here's the thing about this episode Daniel sent us. The framing implies that the software that remembers where you left off in an audiobook is some unsolved, frontier problem. It isn't. It's worse than that. It's a solved problem that everyone solves privately, on their own, every single time.



Herman

Which is a funny thing to say about something as boring as a resume position.



Corn

Daniel's question is this. Playback progress, completion status, listening history. Are those reusable software components, or does every media app reinvent them? And he names some starting points. GStreamer and GstPlay. libmpv. MPRIS on Linux. Media3 and ExoPlayer, and Room, on Android. But he flags it himself, and he's right. Those aren't equivalent technologies. They don't sit at the same level of anything. So the real question is which of them actually remember, and which just hand you the ingredients and walk away.



Herman

That distinction is the whole episode, honestly. Because the moment you lay those five names next to each other, you can see there are three totally different jobs happening. Some of these are event sources. Some are interface standards. One of them persists. And one of them isn't even media software.



Corn

Start with the one that persists, then, because it's the outlier and it's my favorite kind of thing. The one that just did the work while nobody was watching.



Herman

mpv. And it's real. mpv ships a resume feature outright. There's a command called quit-watch-later, bound to Shift+Q by default, and an option, save-position-on-quit. You quit, it stores where you were, and when you open that same file again it resumes from there. That's not a hook or a suggestion. That's the player doing it.



Corn

How does it store it? Because the "how" is where this gets interesting.



Herman

Flat files. It writes what's called a watch-later config file. So it's not a database, there's nothing queryable, you can't ask it "give me everything I've finished this month." It's a file per thing, basically, with your position written in it. And it stores more than position, it'll keep volume, your selected audio and subtitle tracks. There's a watch-later-options setting that controls what gets carried over.



Corn

And the playlist behavior, that's the part I want on the table, because it's small but it's the exact thing every audiobook app gets wrong.



Herman

Right. mpv checks whether any entry in a playlist has a resume config, and if one does, it restarts from there. So if you quit-watch-later on episode five of something, and then later you play the whole show, it doesn't start you at episode one. It starts at five. Which sounds trivial. It is not trivial to build, apparently, because almost nothing else does it.



Corn

So mpv is the answer to Daniel's question, partially. It is a bundled, persistent, playback-tracking feature. It's just locked inside the player.



Herman

That's the catch. It's player-internal. libmpv, the embedding library you'd use to put mpv inside your own app, it exposes the properties you need. There's a time-pos property, there's duration, there are property-change events. But the persistence logic, the actual remembering, lives in the player, not in the library you're embedding. So you don't get to reach in and borrow it cleanly.



Corn

And abrupt termination loses the position.



Herman

Yes, because it writes on an orderly quit. Kill the process and it never got to write.



Corn

So even the one that works has an asterisk on it. Fine. Jump to the one that does the opposite, then. The one that tells you very clearly that remembering is your problem.



Herman

Media3, on Android. And this one's documented, which is why I love it. There's an issue in the androidx media repo. Number two hundred thirty-six, filed at the end of December 2022 by a developer named PaulWoitaschek, and closed in January 2023 by a Google engineer, marcbaechinger. And the maintainer's answer is about as blunt as it gets.



Corn

Give me the answer itself.



Herman

He says the app needs to persist the playlist, the list of media IDs, the current media item, and the position in that media item. And then, on resume, the app re-adds the items to the player, prepares, and starts playback. That's it. That's the guidance. Persist it yourself, hand it back to me, I'll play it.



Corn

The framework is saying "I have no memory. I have reflexes."



Herman

Exactly that. What ExoPlayer gives you is an event stream. The player has four states that come through onPlaybackStateChanged. Idle, buffering, ready, and ended. Ended is the completion signal. And the player exposes a current position. That's a rich set of primitives. It is not a history store.



Corn

So if you're a developer sitting there holding that, you have a number, and you have a state, and you have to decide what to do with them and where to put them.



Herman

Right, and the place you put them is Room. Which is the funny inclusion in Daniel's list, because Room is Android's SQLite layer. It's an object-relational mapper. It knows absolutely nothing about audio. It has no concept of playback. It is a database with a nice API. So you build a table, you give it columns for the media ID and the position, you write to it, you read from it. Every app does this, and every app does it in its own shape.



Corn

Which makes Room the tell, I think. If you're reaching for a general-purpose database to store your playback state, that's the answer to the question. There is no specialized component, so you use the generic one.



Herman

And there's a middle layer on Android that's worth mentioning, because it looks like it does the work and it doesn't. There's a MediaSession callback called onPlaybackResumption. It fires when the service was terminated or the device rebooted, and it's exactly the moment you'd want the framework to hand you back your state. But the documentation says, plainly, your app is responsible for storing the playlist, the metadata, and the start position. It's a prompt, not a store.



Corn

It rings the bell and you have to already know where everything is.



Herman

There's actually something closest to a standard near it, though, and it's the seed of something. Media3 defines two extras keys, a completion status and a completion percentage. Those get used for the resumption notification. So there is, almost, a standardized notion of "how far through is this" on Android.



Corn

Almost.



Herman

An extras key is a string constant. It's a convention about what you put in a bundle for one notification. It's not a schema, and it's not a table, and nothing enforces it.



Corn

So that's two layers down. Events and a hint. Now do GstPlay, because I suspect this is the one that frustrates you.



Herman

It's a high-level playback library, introduced in GStreamer 1.20, built on top of the playbin3 element. The whole point is to make it easy to put playback into an application. And if you look at what it exposes, it is aimed precisely at this problem. There's a get position call that returns your position in nanoseconds. There's get duration, get URI, there's media info. There's a message bus with a whole family of message events, and a signal adapter that turns them into signals.



Corn

Everything you need to build the tracker.



Herman

Everything except the tracker. And here's the detail I'd point at. The documentation warns that the message bus will accumulate messages internally and eventually fill memory unless you consume them. Which is a sentence about a bus. But it is also, if you squint, the entire architecture in one line. It's producing state and throwing it onto a queue, and if nobody's listening, it piles up and then it kills you. There's no resume store. There's no completion flag. And the library's own documentation marks the Play API as considered unstable.



Corn

Unstable. The framework is telling you not to build a product on this surface.



Herman

It's been unstable for a while, and in fairness people ship on it anyway. But it means the high-level convenience layer is not a commitment. If you build your resume logic against it, you have accepted a risk that the app next to you, which talks to the lower level directly, hasn't.



Corn

And I'd flag the years here, because they matter. GStreamer 1.20 was early 2022. This isn't some ancient, unloved corner. It's a current, maintained high-level library, and it still doesn't persist.



Herman

Now MPRIS, because MPRIS is the most interesting of the five for a reason nobody expects.



Corn

It's the standard.



Herman

It's the de facto standard on Linux, and it's a D-Bus specification, current version 2.2, maintained by freedesktop. It defines how a media player announces itself to the rest of the desktop. There's a main interface, a player interface, a track list, a playlists interface. The player interface gives you the position in microseconds, read only. It gives you playback status, playing, paused, stopped. It gives you rate, metadata, and methods to seek, set position, open a URI. And a seeked signal when the position jumps in a way that doesn't match the rate.



Corn

Sounds like it has everything.



Herman

It has everything about right now. It is a live control and query interface. There is no concept in it of a resume position across sessions. There's no "completed versus partially played." There is no history store. And the track list interface is explicit about this. The spec describes it as a short list of tracks recently played or about to be played, intended to give context to the current track, rather than complete access to the player's playlist.



Corn

So it's a window, not a ledger.



Herman

It is a very well specified window. That's the thing. MPRIS standardizes the shape of live playback state beautifully. Ask it where you are in this track right now and any MPRIS-speaking player answers the same way. Ask it what you listened to last week, and it has no idea what you're talking about.



Corn

Herman, this is the pattern, isn't it. We've now walked four things and not one of them remembers anything on your behalf, except mpv, internally, in flat files, on a clean exit.



Herman

Standards exist for control. Not for memory.



Corn

Say that again, slower, because I think that's a finding.



Herman

The standards that exist standardize the live surface, the buttons and the readouts. Nobody has standardized the memory. There is no portable schema for playback history. MPRIS standardizes live shape. Media3 standardizes two extras keys. Neither defines a "here is what a playback record looks like" spec you could carry between players or migrate between frameworks.



Corn

Which means if I listen to half an audiobook in one player and switch to another, that second player has no way to ask the first one where I was. There's no shared vocabulary for that sentence.



Herman

Not through any standard, no. The only reason the two of them could talk is if the developer of both wrote that conversation, and that's app-specific.



Corn

Now, Daniel asked the second-order version of this. Are there specialized libraries? Ready-made playback-history management, recording sessions, resume positions, completed versus partial, persisted between sessions.



Herman

Searches turned up nothing. And I want to be careful here, because "I couldn't find it" is not "it doesn't exist." But this was searched across the usual developer places, and there is no standalone, portable listening-history manager component. What exists are two kinds



Herman (continued)

of thing. There are app-level implementations, the one cited in that issue is an audiobook player called Voice, and its callback is basically a worked example of doing this yourself. And then there are projects like Asuka Player, which advertises persistent playback state, or a thing called playbackEngine, which is a thin wrapper around Media3. Those are applications, or wrappers around applications. They aren't the reusable component.



Corn

I'll take the "couldn't find it" caveat and still be fairly confident, because you can reason about why it's missing. A generic history library has to store a record that means something, and meaning is app-specific. Is "finished" fifteen seconds from the end, or is it the credits, or is it the end state the player reported. Does a podcast episode you abandoned eight minutes in count as partially played or as skipped. Those decisions aren't universal and every app makes them differently.



Herman

And the store is welded to the domain model. An audiobook app already has a chapters table, already has a books table. The resume position is a column on a row it already owns. Asking it to hand that to a foreign library means handing over its data model to something that doesn't know what a chapter is.



Corn

So the reason it's reinvented isn't laziness. It's that the generic version has to be so generic it's barely useful.



Herman

That's the honest structural answer. And the requirements diverge. A music player wants a skip count and a liked flag. An audiobook player wants a position accurate to the second across thirty hours, and cares deeply about chapter boundaries. A podcast app wants to distinguish "heard this" from "saved this." One schema that serves all three is basically "and then some other columns you'll figure out."



Corn

So the answer to Daniel's final question is the disappointing one. No. Playback-state tracking is not an established, reusable cap in the way, say, audio decoding is. Decoding is a solved component. You pull a decoder off the shelf. This one you build in your kitchen



Corn (continued)

every time, and it looks slightly different in every kitchen.



Herman

And there's a wrinkle underneath it that I think is the real story. The framework that should have the strongest opinion here, Media3, is the one most honest about not having one. And look at what the person who filed the issue said. PaulWoitaschek's framing was that media3 seems designed primarily for music playback, with the assumption that the user can freely browse and play individual tracks.



Corn

Which is exactly backwards from the use case that needs this most.



Herman

Long-form audio is the case where resume position is the entire user experience. Nobody abandons a three-minute song and comes back a week later needing to be at two minutes eleven. An audiobook, that's the whole thing. And so the music-centric framework, which assumes you flit between discrete tracks, handles worst the very category that is most dependent on remembering.



Corn

The most demanding use case is the one the assumption ignores.



Herman

There's a service detail that makes it concrete. A Media3 background service will automatically leave the foreground after ten minutes of being paused, stopped, or failed. So your app, if it's behaving, is getting torn down while the user is still nominally in the middle of something. That's not a bug, that's the platform's hygiene. But it means the state has to live somewhere that survives your own process dying, which is exactly why Room is in the picture at all.



Corn

And that's the sentence I'd write down. On Android, playback state is a record that must outlive the player, rebuilt from the app's own database each time the app starts, because the framework won't keep it warm for you.



Herman

There's a small detail in that closed issue that I keep thinking about. A developer in the thread said they do use Media3 with Android Auto, and they have to work around it using the APIs described. So this isn't a rough edge you hit only in a hobby project. It's the supported path. Everybody working around the same thing in public.



Corn

Which brings me back to something, and I'll say it plainly. GStreamer calls its own high-level play API unstable. MPRIS, a community spec with no company behind it, is the thing everyone targets and it's rock solid. And neither of them persists a byte of history. The reliability and the persistence are totally uncorrelated. The most solid standard in the room remembers nothing.



Herman

And the one thing that persists, mpv, persists in flat text files, per file, written on a clean exit.



Corn

So the landscape is, events at the bottom with no memory, standards in the middle for the live view, one player doing its own thing internally in files, and then every app rolling its own table on a generic database. Five things Daniel named, and not one of them is the component he was asking about.



Herman

The component he was asking about doesn't have a name. That's the finding. There's no artifact to point at. It's a hole in the stack that everyone fills with their own material.



Corn

Somebody's listening to this thinking, "well, I just want a podcast app to remember where I was, why is this hard." And the answer is it isn't hard, it's just unowned. It's nobody's job. The framework provides the instruments, the standard provides the display, and the app provides the memory, and the app has to do that in its own dialect.



Hilbert

So my cousin ran an AM station in upstate New York and the whole automation was on paper tape.



Herman

Sorry, the what now.



Hilbert

Paper tape. Punched tape, the width of your thumb, running through a reader next to the cart machine. He had it built in seventy-four. It kept the log. Which record, what time, and whether it went the whole way or you had a skip.



Corn

If the needle skipped?



Hilbert

It put a hole for it. One hole past the runout groove was a clean play, two holes was a partial. The next guy coming on shift could look at the tape and see the whole night in front of him without asking anybody. And if a record got interrupted, the tape marked the groove, so whoever came in next dropped the needle at the right spot. They had resume before anybody had a screen.



Herman

How did it handle one record out of a stack, though? If six records went by and the tape only logged one...



Hilbert

There was a card in the drawer for each record. Tape said which sleeve to pull. That was Rita's job, and Rita was fast. Twenty-two seconds from the tape stopping to the cart being swapped, and she did that for eleven years.



Corn

Twenty-two seconds is a very specific number.



Hilbert

I timed her. Nobody asked me to. She never lost a playback slot in eleven years either. That tape was mechanical. Power went out, tape didn't care. It still knew.



Herman

Wait, so a power failure in the middle of a record...



Hilbert

The reader held its place. The tape had the holes already punched. You lose power for forty minutes, tape still says what got played and what didn't.



Corn

Then, presumably, it also fetched your groceries.



Hilbert

He sold the design to NASA. Fell through. The astronauts said the tape was too loud in a capsule.



Corn

Sure.



Herman

Did he ever...



Hilbert

The paper was his own mix. Nobody wrote the formula down. He thought people were after it. So it's gone.



Corn

He does this every week, Herman.



Hilbert

Anyway, they had the resume bit working before any of these frameworks existed. People forget that.



Corn

The tape's gone, and the formula's gone, and the system is gone, and yet here we are in the same spot. Every app writing its own little watch-later config, and calling it a feature.



Herman

The clean play versus the partial. Two holes in a strip of paper. That's a completion status. That's the thing Media3 hands you as two strings in a bundle, forty years later, as a convention you have to implement yourself.



Corn

Daniel's answer, and I want to close on this because it's cleaner than I expected it to be. No. There is no reusable playback-history component waiting for you. What exists is a decoder layer that doesn't remember, a standard that only knows right now, one player that keeps its own notes in text files, and a database library that has never heard of audio in its life. Everything else is the app's job.



Herman

The open question, the one I'd leave in the air, is why. My best answer is that the data model is too app-specific to standardize, and everybody assumes it's trivial until they try to write it. But I don't fully buy my own answer, because the paper tape suggests a station solved it with holes and a person who timed things.



Corn

Because the requirement isn't exotic. It's just unowned. That's the part I'd watch going forward. As long-form audio keeps growing, the gap between what the frameworks assume and what listeners expect is only getting wider. At some point the pressure either produces a standard or it produces another twenty apps each with their own tiny table. And I know which one my money's on.



Herman

If this was your kind of episode, go back for episode twenty-two twenty, When Home Assistant Breaks Your Audio. Thanks to our producer, Hilbert Flumingtop. This has been My Weird Prompts.



Corn

If you got something out of this one, a rating and a review helps more than you'd think. Send us your own prompt on Telegram at t dot me slash MWP listener bot.



Herman

We'll be back soon.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **GstPlay [primary]**
   GStreamer docs
   https://gstreamer.freedesktop.org/documentation/play/gstplay.html

2. **Play Library index (API unstable) [primary]**
   GStreamer
   https://gstreamer.freedesktop.org/documentation/play/index.html

3. **MPRIS D-Bus Interface Specification v2.2 [primary]**
   freedesktop
   https://specifications.freedesktop.org/mpris/latest/

4. **MediaPlayer2.Player interface [primary]**
   MPRIS v2.2
   https://specifications.freedesktop.org/mpris/latest/Player_Interface.html

5. **MediaPlayer2.TrackList interface [primary]**
   MPRIS v2.2
   https://specifications.freedesktop.org/mpris/latest/Track_List_Interface.html

6. **mpv manual (master) [primary]**
   RESUMING PLAYBACK, watch history, libmpv
   https://mpv.io/manual/master/

7. **libmpv client.h (master, API v2.3) [primary]**
   mpv-player/mpv
   https://github.com/mpv-player/mpv/blob/master/libmpv/client.h

8. **androidx/media issue #236 Guidance on Playback Position persistance [primary]**
   https://github.com/androidx/media/issues/236

9. **Background playback with a MediaSessionService (last updated 2026-10-01) [primary]**
   Android Developers
   https://developer.android.com/media/media3/session/background-playback

10. **Resuming playback**
    mpv manual mirror
    https://rksvc.github.io/mpv-manual/resuming-playback/

11. **Player events**
    Android Developers
    https://developer.android.com/media/media3/exoplayer/listening-to-player-events

12. **Control and advertise playback using a MediaSession**
Android Developers
https://developer.android.com/media/media3/session/control-playback