



Parsing Browser Session Files in Rust

Different browsers store their session (open tabs/windows) in different formats and locations. Below, we provide Rust code (with minimal dependencies like `serde`, `serde_json`, `plist`, and optionally `rusqlite`/`lz4_flex`) to handle Chrome/Brave/Arc (Chromium-based browsers), Safari, and Firefox on macOS (with notes for other OS). The code extracts each open tab's URL and title, and identifies the active tab in each window whenever possible. You can then log this information and use a file watcher to update the log whenever session files change.

Safari (macOS)

Safari's session storage depends on the version:

- **Safari 14 and earlier:** Stored in `~/Library/Safari/LastSession.plist` (binary plist). This plist contains an array `"SessionWindows"` of window records, each with a `"TabStates"` array of tabs. Each tab entry has keys like `"TabURL"` and `"TabTitle"` for the last viewed page ¹.
- **Safari 15 and later:** Uses a SQLite database `SafariTabs.db` in the Safari container (e.g. `~/Library/Containers/com.apple.Safari/Data/Library/Safari/SafariTabs.db`) ². The `windows` table has `is_last_session` flag and an `active_tab_group_id` linking to the `bookmarks` table, which holds records of open tabs (with URL and title).

Below is code to handle both cases. We use the `plist` crate to parse the older plist format, and `rusqlite` to query the newer database. (Feel free to add proper error handling as needed.)

```
use plist::Value;
use rusqlite::{Connection, NO_PARAMS};
use std::fs;

fn parse_safari_session() -> Result<(), Box
```

```

let window_rows = win_stmt.query_map(NO_PARAMS, |row| {
    Ok((row.get::<_, i64>(0)?, row.get::<_, i64>(1)?))
})?;
for win_res in window_rows {
    let (window_id, group_id) = win_res?;
    println!("Safari Window {}:{}", window_id);
    // Query all tabs (bookmarks) in this window's active tab group
    let mut tab_stmt = conn.prepare(
        "SELECT title, url FROM bookmarks WHERE parent = ? ORDER BY
order_index"
    )?;
    let tabs = tab_stmt.query_map([group_id], |row| {
        Ok((row.get::<_, Option<String>>(0)?, row.get::<_, Option<String>>(1)?))
    })?;
    let mut index = 0;
    for tab_res in tabs {
        index += 1;
        let (title, url) = tab_res?;
        let title_str = title.as_deref().unwrap_or("<No Title>");
        let url_str = url.as_deref().unwrap_or("<No URL>");
        // (Active tab identification not readily available from
        SafariTabs.db)
        println!("    Tab {}: \"{}\" - {}", index, title_str, url_str);
    }
}
} else {
    // Safari 14 or earlier: parse LastSession.plist if it exists
    let plist_path = dirs::home_dir().unwrap().join("Library/Safari/
LastSession.plist");
    if plist_path.exists() {
        let data = fs::read(plist_path)?;
        let plist = Value::from_reader_xml(data.as_slice())?; // parse
binary plist
        if let Value::Dictionary(root) = plist {
            if let Some(Value::Array(session_windows)) =
root.get("SessionWindows") {
                for (w_idx, win_val) in session_windows.iter().enumerate() {
                    if let Value::Dictionary(win_dict) = win_val {
                        println!("Safari Window {}:{}", w_idx + 1);
                        if let Some(Value::Array(tab_states)) =
win_dict.get("TabStates") {
                            for (t_idx, tab_val) in
tab_states.iter().enumerate() {
                                if let Value::Dictionary(tab_dict) =
tab_val {
                                    let url = tab_dict.get("TabURL")
                                        .and_then(|v| v.as_string())

```

Notes: Safari's LastSession.plist stores each tab's last viewed URL and title explicitly (making parsing easy) . In Safari 15+, we retrieve tabs via the bookmarks table. (The active tab in each window is typically the one visible when the session was saved, but Safari's database doesn't expose a simple "active tab index" field in an obvious way, so all open tabs are listed without highlighting one as active.)

Chrome / Brave / Arc (Chromium-based)

Chrome and other Chromium-based browsers (e.g. Brave, Arc) save session information in a custom binary format often referred to as **SNSS** (Session Storage) files ⁵. These files are located in the browser's profile directory, typically under a "Sessions" subfolder:

- **Chrome (macOS):** `~/Library/Application Support/Google/Chrome/Default/Sessions/`
 - **Brave (macOS):** `~/Library/Application Support/BraveSoftware/Brave-Browser/Default/Sessions/`
 - **Arc (macOS):** `~/Library/Application Support/Arc/User Data/Default/Sessions/` (likely similar structure as Chrome)

On Windows and Linux, paths are analogous in the Chrome/Brave user data directories [6](#) [7](#). The Sessions folder contains pairs of files (e.g. `Session_*` and `Tabs_*`) representing the last or current session data [8](#) [9](#).

The SNSS format starts with a 4-byte signature "SNSS" and a 4-byte version (currently 1). It then consists of a sequence of records: each record begins with a 2-byte little-endian length (excluding those two bytes),

an 1-byte command ID, and then the command-specific payload ⁵. Using Chromium's source, we know some relevant command IDs ¹⁰ ¹¹:

- **0** - **SetTabWindow**: associates a tab ID with a window ID.
- **2** - **SetTabIndexInWindow**: order of tabs in a window.
- **6** - **UpdateTabNavigation**: contains a navigation entry (URL, title, etc.) for a tab ¹². Multiple of these occur per tab (one per history entry).
- **7** - **SetSelectedNavigationIndex**: which navigation entry is current in a tab.
- **8** - **SetSelectedTabInIndex**: which tab index is active in a window.

Using these, we can parse the session files to reconstruct open windows and tabs. Below is a Rust function that opens the most recent **Session** and **Tabs** files and parses them to list all open tabs with their titles and URLs, marking the active tab in each window. (For simplicity, we pick the latest files by sorting filenames; in a real scenario, you might choose **Current Session**, **Current Tabs**, or **Last Session** files explicitly.)

```
use std::fs::File;
use std::io::{Read, Cursor};
use byteorder::{LittleEndian, ReadBytesExt};

fn parse_chromium_session(session_dir: &str) -> std::io::Result<()> {
    // Find latest Session and Tabs files in the given directory
    let mut session_file = None;
    let mut tabs_file = None;
    if let Ok(entries) = fs::read_dir(session_dir) {
        for entry in entries {
            let path = entry?.path();
            if let Some(name) = path.file_name().and_then(|n| n.to_str()) {
                if name.starts_with("Session_") {
                    // assuming lexicographically largest is latest
                    if session_file.as_ref().map_or(true, |f: &String| name > f) {
                        session_file = Some(path.display().to_string());
                    }
                } else if name.starts_with("Tabs_") {
                    if tabs_file.as_ref().map_or(true, |f: &String| name > f) {
                        tabs_file = Some(path.display().to_string());
                    }
                }
            }
        }
    }
    let session_path = session_file.expect("No Session file found");
    let tabs_path = tabs_file.expect("No Tabs file found");
    let mut data = Vec::new();
    File::open(&session_path)?.read_to_end(&mut data)?;
    File::open(&tabs_path)?.read_to_end(&mut
```

```

data)?; // concatenate Tabs data as well
// Data now contains the concatenated records of Session and Tabs
let mut cur = Cursor::new(&data);
// Verify header "SNSS"
let mut signature = [0u8; 4];
cur.read_exact(&mut signature)?;
if &signature != b"SNSS" {
    eprintln!("Invalid session file signature");
    return Ok(());
}
let _version = cur.read_u32::<LittleEndian>()?;
// version (should be 1)
// Structures to hold parsed info
use std::collections::{HashMap, HashSet};
struct TabData { url: String, title: String, active_nav_index: u32 }
let mut windows: HashMap<u32, Vec<u32>> = HashMap::new();           //
window_id -> list of tab_ids
let mut tab_to_index: HashMap<u32, u32> = HashMap::new();           // tab_id -
> index in window
let mut tab_data: HashMap<u32, TabData> = HashMap::new();           // tab_id -
> current URL/title
let mut window_active_tab: HashMap<u32, u32> = HashMap::new();       //
window_id -> active tab index
// Parse all records
while cur.position() < data.len() as u64 {
    // Each record: 2-byte length, then that many bytes: 1 byte id + payload
    if cur.position() == data.len() as u64 { break; }
    let record_len = match cur.read_u16::<LittleEndian>() {
        Ok(len) => len,
        Err(_) => break
    };
    if record_len == 0 { continue; }
    let id = cur.read_u8()?;
    let payload_len = record_len as u64 - 1;
    let payload_start = cur.position();
    // Helper to ensure we consume the payload
    macro_rules! finish_record { () => {
        let new_pos = payload_start + payload_len;
        cur.set_position(new_pos);
    } }
    match id {
        0 => { // SetTabWindow: payload likely [TabID:u32, WindowID:u32]
            let tab_id = cur.read_u32::<LittleEndian>()?;
            let window_id = cur.read_u32::<LittleEndian>()?;
            windows.entry(window_id).or_default().push(tab_id);
        },
        2 => { // SetTabIndexInWindow: [TabID:u32, Index:u32]
            let tab_id = cur.read_u32::<LittleEndian>()?;
            let index = cur.read_u32::<LittleEndian>()?;
        }
    }
}

```

```

        tab_to_index.insert(tab_id, index);
    },
6 => { // UpdateTabNavigation: [TabID:u32, NavIndex:u32,
URL_len+URL, Title_len+Title, ...]
    let tab_id = cur.read_u32::<LittleEndian>()?;
    let nav_index = cur.read_u32::<LittleEndian>()?;
    // Read URL (ASCII string prefixed by 32-bit length)
    let url_len = cur.read_u32::<LittleEndian>()? as usize;
    let mut url_buf = vec![0; url_len];
    cur.read_exact(&mut url_buf)?;
    let url = String::from_utf8_lossy(&url_buf).into_owned();
    // Read Title (UTF-16 string prefixed by 32-bit length of
characters)
    let title_chars = cur.read_u32::<LittleEndian>()? as usize;
    let mut title_buf = vec![0; title_chars * 2];
    cur.read_exact(&mut title_buf)?;
    // Convert UTF-16 little-endian bytes to Rust String
    let title = String::from_utf16_lossy(
        bytemuck::cast_slice(&title_buf)
    );
    // We skip the rest of the fields in this record for brevity
    // Assume this command is for the current navigation entry
    tab_data.entry(tab_id).or_insert(TabData {
        url: url.clone(), title: title.clone(), active_nav_index: 0
    });
    // (We don't know yet if this is the active entry; see command
7)
},
7 => { // SetSelectedNavigationIndex: [TabID:u32,
SelectedNavIndex:u32]
    let tab_id = cur.read_u32::<LittleEndian>()?;
    let sel_index = cur.read_u32::<LittleEndian>()?;
    if let Some(tab) = tab_data.get_mut(&tab_id) {
        tab.active_nav_index = sel_index;
    }
},
8 => { // SetSelectedTabInIndex: [WindowID:u32,
SelectedTabIndex:u32]
    let window_id = cur.read_u32::<LittleEndian>()?;
    let sel_tab_index = cur.read_u32::<LittleEndian>()?;
    window_active_tab.insert(window_id, sel_tab_index);
},
_ => {
    // For other command IDs, skip their payload
}
}
finish_record!();
}

```

```

// Now output the gathered data
for (&win_id, tab_ids) in &windows {
    println!("Chromium Window {}:", win_id);
    // Sort tabs by their index within the window (to list in order)
    let mut sorted_tabs = tab_ids.clone();
    sorted_tabs.sort_by_key(|tid|
        tab_to_index.get(tid).copied().unwrap_or(0));
    for &tab_id in &sorted_tabs {
        let title = tab_data.get(&tab_id).map(|d|
            d.title.as_str().unwrap_or("<No Title>"));
        let url = tab_data.get(&tab_id).map(|d|
            d.url.as_str().unwrap_or("<No URL>"));
        let tab_index = tab_to_index.get(&tab_id).copied().unwrap_or(0);
        let active_index =
            window_active_tab.get(&win_id).copied().unwrap_or_default();
        let is_active = active_index == tab_index;
        let mark = if is_active { "*" } else { " " };
        println!("{}Tab {}:{} - {}", mark, tab_index, title, url);
    }
}
Ok(())
}

```

In this code, we read both the latest `Session_...` and `Tabs_...` files (concatenating their contents) – these files contain interleaved SNSS records describing the session [8](#) [9](#). We then iterate through the records, interpreting known command IDs to build our data structures. Notably, for each window we identify the active tab via the `SetSelectedTabInIndex` (ID 8) command, and mark it with `*` in the output. Each tab's current URL and title come from the last `UpdateTabNavigation` (ID 6) record (paired with a `SetSelectedNavigationIndex` to confirm which entry is active) [13](#) [14](#).

Note: This parsing is based on Chrome's session format as documented in Chromium's code and forensic research [5](#) [10](#). Arc Browser uses the Chromium engine (as of 2025) and stores sessions in the same format (users have reported Arc's “Tabs session (SNSS) files” can be parsed with similar tools [15](#)). Brave is also Chromium-based and uses the same session file structure in its profile directory.

Firefox

Firefox stores open tabs in a JSON-based session file which is compressed with Mozilla's LZ4 (with a custom header). On macOS, the session files reside in your Firefox profile directory, e.g. `~/Library/Application Support/Firefox/Profiles/<Profile>/sessionstore-backups/`. The **current** session (when Firefox is running) is in `recovery.jsonlz4` (with a backup in `recovery.baklz4`), and the last closed session is saved to `sessionstore.jsonlz4` when Firefox exits [16](#) [17](#). We'll attempt to read `recovery.jsonlz4` if available (for live state), otherwise fallback to `sessionstore.jsonlz4` for the last session.

The `.jsonlz4` format starts with `"mozLz40\0"` and 4 bytes of uncompressed JSON size ¹⁸ ¹⁹. We can skip the 8-byte header and use an LZ4 decoder to get the JSON text ²⁰. The JSON contains an array `"windows"` of window objects ²¹. Each window has a `"tabs"` array, and a `"selected"` field indicating the 1-based index of the active tab in that window ¹⁴. Each tab has an `"entries"` array (each entry has a URL and title) and an `"index"` field (1-based) pointing to the current entry in that tab's history ¹⁴. We'll extract each window's tabs and mark the active tab.

Dependencies: we use `lz4_flex` for decompression and `serde_json` for parsing JSON into our structs.

```
use lz4_flex::decompress_size_prepended;
use serde::Deserialize;
use std::fs::File;
use std::io::Read;
use std::path::Path;

#[derive(Deserialize)]
struct SessionEntry {
    url: String,
    title: Option<String>,
}

#[derive(Deserialize)]
struct SessionTab {
    entries: Vec<SessionEntry>,
    index: u32, // 1-based index of current entry
    #[serde(default)] pinned: bool,
}

#[derive(Deserialize)]
struct SessionWindow {
    tabs: Vec<SessionTab>,
    selected: u32, // 1-based index of active tab
}

#[derive(Deserialize)]
struct SessionStore {
    windows: Vec<SessionWindow>,
}

fn parse_firefox_session(profile_path: &Path) -> Result<(), Box<dyn std::error::Error>> {
    // Prefer recovery.jsonlz4 if it exists (browser is running), else sessionstore.jsonlz4
    let main_path = if profile_path.join("sessionstore-backups/recovery.jsonlz4").exists() {
        profile_path.join("sessionstore-backups/recovery.jsonlz4")
    } else {
        profile_path.join("sessionstore.jsonlz4")
    };
}
```

```

let mut file = File::open(&main_path)?;
// Read and skip "mozLz4\0" (8 bytes), then decompress the rest
let mut header = [0; 8];
file.read_exact(&mut header)?;
if &header[0..6] != b"mozLz4" {
    return Err("Not a Firefox JSONLZ4 file".into());
}
// The rest of the file: first 4 bytes of this segment are the uncompressed
size (LE32), which `decompress_size_prepended` will use
let mut compressed = Vec::new();
file.read_to_end(&mut compressed)?;
let decompressed = decompress_size_prepended(&compressed)?; // requires
lz4_flex = "0.9"
// Parse JSON
let session: SessionStore = serde_json::from_slice(&decompressed)?;
for (w_idx, window) in session.windows.iter().enumerate() {
    println!("Firefox Window {}:", w_idx + 1);
    for (t_idx, tab) in window.tabs.iter().enumerate() {
        // Get current entry for the tab (index is 1-based in JSON)
        let entry_index = (tab.index as usize).saturating_sub(1);
        let entry = tab.entries.get(entry_index);
        let url = entry.map(|e| e.url.as_str()).unwrap_or("<no url>");
        let title = entry.and_then(|e| e.title.as_deref()).unwrap_or("<no
title>");
        let mark = if window.selected == (t_idx + 1) as u32 { "*" } else {
            " "
        };
        let pin = if tab.pinned { "(pinned)" } else { "" };
        println!("{}Tab {}: \"{}\" - {} {}", mark, t_idx + 1, title,
url, pin);
    }
}
return Ok(());
}

```

This code finds the Firefox profile's session file, decompresses it, and uses Serde to deserialize the JSON into our `SessionStore` struct. We then iterate through each window and tab. We identify the active tab using the window's `selected` index (comparing to the tab's position)¹⁴, and fetch the tab's current URL/title using its `index` into the `entries` list. In the output, the active tab is marked with `*`. We also show if a tab is pinned (Firefox's JSON has a `pinned` flag on tabs, defaulting to false).

Note: You may need to determine the correct profile path. In a multi-profile setup, the default profile often has "default-release" in its folder name. In a real tool, you could read `profiles.ini` to find the default profile directory²². Here, for simplicity, supply the profile path (or auto-detect the first profile directory).

Using a File Watcher

Once you have functions to parse each browser's session data, you can set up a loop or file-system watcher to monitor changes. For example, using the `notify` crate, you could watch the specific session files (e.g., `SafariTabs.db`, the `Chrome Sessions` folder, or `Firefox's recovery.jsonlz4`) and call the parsing functions whenever they change. This way, the log of open tabs updates in near real-time as the browser session updates.

Sources:

- Safari session storage in `LastSession.plist` (pre-v15) and example keys (`SessionWindows`, `TabStates`, `TabURL`, `TabTitle`) 3 4 1 . Change in Safari 15+ to `SafariTabs.db` 2 .
 - Chrome/Chromium session files location and SNSS format 7 5 . Chromium session command IDs and structure 10 23 12 .
 - Firefox sessionstore format and usage of LZ4 with header "mozLz4" 18 20 . JSON structure for windows/tabs (`selected`, `entries`, etc.) 21 14 .
-

1 Safari Forensic | David Koepi

<https://davidkoepi.wordpress.com/2013/04/20/safariforensic/>

2 Safari 15 does not already use `LastSession.plist` · Issue #78 · ydkhatri/mac_apt · GitHub

https://github.com/ydkhatri/mac_apt/issues/78

3 4 Rescuing open windows and tabs from Safari - Apple Community

<https://discussions.apple.com/thread/252894091>

5 9 12 13 Chrome Session and Tabs Files (and the puzzle of the pickle) | Digital Investigation

<https://digitalinvestigation.wordpress.com/2012/09/03/chrome-session-and-tabs-files-and-the-puzzle-of-the-pickle/>

6 7 Google Chrome History Location | Chrome History Viewer

<https://www.foxtonforensics.com/browser-history-examiner/chrome-history-location>

8 How do I recover tab / session information from Chrome / Chromium?

<https://superuser.com/questions/662329/how-do-i-recover-tab-session-information-from-chrome-chromium>

10 11 23 chrome/browser/sessions/session_service.cc - Issue 9969104: Make session service support migration between 32bit and 64bit - Code Review

https://chromiumcodereview.appspot.com/9969104/diff/2002/chrome/browser/sessions/session_service.cc?context=10&column_width=80&tab_spaces=8

14 command line - Output URL of open firefox tabs in terminal - Ask Ubuntu

<https://askubuntu.com/questions/338294/output-url-of-open-firefox-tabs-in-terminal>

15 Data Lost After Creating Account on Other Mac : r/ArcBrowser - Reddit

https://www.reddit.com/r/ArcBrowser/comments/16cj5wb/data_lost_after_creating_account_on_other_mac/

16 17 Analysing Firefox Session Restore data | Foxton Forensics

<https://www.foxtonforensics.com/blog/post/analysing-firefox-session-restore-data-mozlz4-jsonlz4>

18 19 20 21 22 Decoding Firefox session store data
<https://blog.dend.ro/decoding-firefox-session-store-data/>