ChatGPT

Star Wars: The Old Republic Combat Log Technical Reference

Overview of Combat Log Format

In Star Wars: The Old Republic (SWTOR), enabling combat logging causes the game to record a timestamped text entry for every combat-related action. Each line in the combat log follows a structured syntax capturing **when** an event happened, **who** was involved (source and target), **what** ability or effect was used, and the **result** (damage, healing, etc.), along with position, health, and threat details [1] [2]. At a high level, a single combat log line can be represented as:

```
[timestamp] [source] [target] [ability] [event: effect] (value) <threat>
```

For example, a real combat log entry might look like this (split across lines for readability):

```
[01:28:40.284]
[@Pugzeroone#688438700531589|(246.10,-141.57,-232.11,-88.18)|(450254/450254)]
[Lord Kanoth {4494876548792320}:11760001999148|(260.00,-144.00,-232.06,116.61)|
(43894876/43909316)]
[Ion Cell {3963919806758912}]
[ApplyEffect {836045448945477}: Shocked {3963919806759210}] (937 ~936 energy
{836045448940874}) <2811>
```

This entry (taken from an Operation combat log) is packed with information. In the sections below, we break down each component of such a line in detail.

Timestamp

Each log line begins with a timestamp in square brackets. The timestamp format is `HH:MM:SS.mmm` (hours, minutes, seconds, and milliseconds) in 24-hour time. For example, `[01:28:40.284]` indicates the event occurred at 1:28 AM plus 0.284 seconds. The game logs events with millisecond precision, which means multiple events can share the **exact same timestamp** if they occurred in the same millisecond. (In the example above, several entries share `01:28:40.284` or `01:28:40.285` timestamps, indicating simultaneous sub-events.)

Date: The log file's name encodes the date and start time of the session (e.g. `combat_2023-09-25_01_14_15_971249.txt` for a session on 2023-09-25 starting at 01:14:15). The timestamp in each line, however, includes only the time of day. If a play session spans midnight or multiple days, SWTOR starts a new log file, so the date is implicit from the file name rather than printed in each line.

In older versions of SWTOR (pre-2.0), timestamps were logged only to the second [3], but modern logs always include milliseconds for higher resolution.

Source and Target Fields

Following the timestamp, the next two bracketed fields are the **source** and **target** of the event. These fields encode *who* performed the action and *who* it was performed on. The source and target fields have an identical structure, which packs several sub-fields together separated by `|` (pipe) characters:

```
[name identifier | (x,y,z,orientation) | (currentHealth/maxHealth)]
```

Each of these subcomponents is explained below:

- **Name Identifier:** This part identifies the entity (player, NPC, companion, etc.) by name and unique identifiers. Players are identified by an `@` prefix plus their character name and a numeric hash, while NPCs and companions are identified by their name plus numeric IDs in braces and after a colon. For example:
- `@Pugzeroone#688438700531589` – a player character named “Pugzeroone” with a unique ID `688438700531589`. The `@` indicates a player, and the `#...` number is a unique identifier for that character (unique within the log, possibly an internal unique ID for the session or account) introduced in patch 7.0. This helps distinguish players with the same name and ties events to a specific character [4] [5].

- `Lord Kanoth {4494876548792320}:11760001999148` – a non-player character (NPC) named "Lord Kanoth". The number in `{...}` is a static *String ID* for that name or NPC type, and the number after the colon is a unique instance identifier for that specific NPC spawn. In this case, `4494876548792320` is the static ID for the boss Lord Kanoth (consistent across all logs and languages), and `11760001999148` is the unique spawn ID in that operation instance [6] [7]. Every NPC or object is logged with the format `Name {StaticID}:InstanceID`. This convention was added to SWTOR’s logs in **Game Update 7.0** and allows distinguishing multiple entities with the same name. (Prior to 7.0, combat logs only showed the static ID in braces for NPCs and did not differentiate instances.)

- **Coordinates:** After the name/ID, the source/target field includes a parenthesized 3D coordinate and facing angle: `(x, y, z, orientation)`. These are the position of the entity in the game world at the time of the event. For example, `(-1.70,-74.06,39.04,74.57)` represents an entity’s location on the map (X, Y, Z coordinates) and a facing of 74.57 degrees. Coordinates are measured in the game’s internal coordinate system (units are typically meters; the origin and axes vary by planet/zone). The **orientation** is likely in degrees (0–360) or radians indicating which direction the entity is facing [8]. This coordinate logging was **introduced in 7.0**, enabling new analysis features like positional heatmaps and fight replays (e.g. Parsely’s tactical replays use these to animate movements) [9]. Prior to 7.x, combat logs did not include positional data, so this was a major enhancement.

- **Health:** The final part of the source/target field is the current and maximum health of that entity at the moment of the event, in the format `(currentHP/maxHP)`. For example, `(43894876/43909316)` indicates the target currently has 43,894,876 HP out of 43,909,316 maximum. These values are updated as damage and healing occur. Generally, the health shown is *after* the event's effect is applied (so a damage event will typically show the target's health post-damage). For instance, immediately after a damaging hit on Lord Kanoth, his health decreased from `43894876` to `43885136` in the subsequent log entry [10] [11]. Health values allow parsers to determine how much damage remains to kill a target or how much overheal occurred, and can also signal when an entity is defeated (current HP going to 0).

Special cases for Source/Target: In some log lines, the target or even the source might be "empty" or denote the same entity:

- If an event has **no target** (e.g. an area trigger or environment event), the target field will be empty `[]`. For example, when a player enters a new zone, an `AreaEntered` event is logged with no target:

```
[@Pugzeroone#...|(...)|(...)] [] [] [AreaEntered {...}: Nar Shaddaa {...}]
(he3001) <v7.0.0b>
```

  Here the target field is `[]` since this event isn't directed at a creature or player—Nar Shaddaa is an environment zone event [12].

- If the target is the **same as the source** (self-targeting), the target field is abbreviated as `[=]` (an equals sign) [13]. This shorthand means the target entity is identical to the source entity, avoiding repetition. For example, when a player applies a buff or heal to themselves, the log might show `[SourcePlayer] [=] [Ability] [ApplyEffect: Buff...] ...`, using `[=]` in place of repeating the player's name as target.

- The source field can also use `[=]` in rare cases (like certain Death events or environmental effects) to indicate the source is the same as target or unknown. An example from the logs is a line where a player dies and the source is `[=]` with an empty ability, indicating no external source caused the death [14].

- If a companion, pet, or object is acting on behalf of a player, the source name can be a composite of owner and pet. For example, `@Pugzeroone#.../Z0-0M {3841216886079488}:143123437359` might appear as a source, indicating the companion **Z0-0M** (with its static and instance IDs) belonging to player Pugzeroone. This format "Owner/Minion" denotes that the action was performed by the minion (Z0-0M), but is associated with the owner (Pugzeroone) [15]. In the logs, this was observed when applying buffs that originate from a companion or legacy perk – the companion's name appears after the slash. (Patch 7.0 enabled such cross-faction buff applications via companions, causing both Republic and Imperial buff versions to be applied, sometimes attributed to a companion to distinguish faction-specific effects.)

String IDs: The numeric IDs in braces (e.g. `{836045448945477}`) are consistent identifiers for abilities, effects, and names. BioWare included these so that logs could be compared across different language clients [2]. For example, "ApplyEffect" has an ID `{836045448945477}` that will be the same in French or German logs, and the Nar Shaddaa planet has ID `{137438987989}` regardless of locale. This is extremely useful for parsers – instead of matching text which might be localized, one can match these IDs. A **unique ID** after a colon (like `:11760001999148`) is not a static identifier but an ephemeral unique instance ID (often for NPCs or temporary objects), used to differentiate multiple copies of the same entity during that session.

Ability and Effect Fields

After source and target, the next bracketed segment is the **ability** used, and following that is the **action/event** that occurred. These two fields together describe *what happened*.

- **Ability Name and ID:** The ability field is written as `[AbilityName {abilityID}]`. For example, `[Hammer Shot {801299163512832}]` indicates the **Hammer Shot** ability (with its unique identifier). If the event isn't directly tied to a named ability (for instance, a passive effect or no ability at all), this field can be empty `[]`. An empty ability field usually pairs with a generic event like a simple `Event` (see below). In some cases, the ability field might contain an item or effect name – e.g. a stim use is logged with the stim's name as the "ability" triggering an effect [16]. The ability ID in braces is a static reference to that ability.

Example: In the log snippet below, the player's ability *Hammer Shot* is logged as the ability field, and *Shocked* (from Ion Cell) appears as the effect:

```
[@Pugzeroone...|(450254/450254)] [Lord Kanoth...|(43894876/43909316)]
[Ion Cell {3963919806758912}] [ApplyEffect: Shocked {3963919806759210}] ...
```

Here `Ion Cell` is a passive effect (the Ion Cell tactical or discipline ability) that caused the *Shocked* effect on the target. So `Ion Cell` is logged as the "ability" that led to an ApplyEffect.

- **Event Type and Effect Name:** Next is the `[EventType {ID}: EffectName {ID}]` field. This describes the action or effect that took place. Common values for the event type include:
- **ApplyEffect {836045448945477}:** This means an effect was applied – it could be damage, healing, a buff, a debuff, etc. The specific effect name that was applied follows after the colon. For example, `ApplyEffect {836045448945477}: Damage {836045448945501}` is used for a damage event [17], and `ApplyEffect {...}: Heal {836045448945500}` for a heal [18]. Likewise, applying a buff or debuff shows up as `ApplyEffect: <BuffName>`.
- **RemoveEffect {836045448945478}:** This indicates an effect ended or was removed. For example, `RemoveEffect {...}: Sprint {810670782152704}` means the Sprint effect was removed (perhaps the player toggled sprint off or it was auto-removed) [19]. Buffs expiring or being cleansed are recorded with RemoveEffect.

- **Event {836045448945472}:** A generic event occurred. The specific event name after the colon clarifies what it was. Examples include `Event: AbilityActivate {836045448945479}` when an ability begins activation (e.g. the player started casting or using an ability) [20], `Event:`

`AbilityDeactivate {836045448945480}` when an ability or channel ends, `Event: Death {836045448945493}` when an entity dies [21], and `Event: TargetSet {836045448953668}` when a character changes targets [22] [23]. The *TargetSet* event is noteworthy: starting in 7.0, the log even tracks when group members switch target or focus. For instance, one log line shows `[@Viszlo...][@Menkey...] [] [Event: TargetSet]` meaning Viszlo switched their target to Menkey at that moment [22]. These granular events (target changes, etc.) were not present in early versions of the combat log – they came along with the expanded group-combat logging in 7.x.

- **Other event types:** There are additional event keywords like `Event: EnterCombat` and `Event: ExitCombat` (not shown above, but these exist to mark when an entity enters or leaves combat state), `Event: ChallengeBegin` (for start of a ranked PvP match, etc.), and more. There are also special cases such as `DisciplineChanged` events when a player switches their Combat Style or Discipline mid-session (a 7.0 feature). For example, when our example player switched from Vanguard/Plasmatech to Vanguard/Shield Specialist, the log recorded `DisciplineChanged {836045448953665}: Vanguard {...}/Shield Specialist {...}` [24]. This event has no ability (so ability field is `[]`) since it's not triggered by a combat ability but by a loadout change.

Each event type and effect name has an ID in braces, similar to abilities, which can be cross-referenced. **Damage** is always indicated by the effect name "Damage {836045448945501}" with an ApplyEffect event [17]. **Healing** likewise uses "Heal {836045448945500}". Most common combat effect and event IDs are consistent (836045448945477 for ApplyEffect, 478 for RemoveEffect, etc., as seen in every line).

Value: Damage, Healing, and Mitigation Details

The portion in parentheses `( ... )` after the event/effect describes the **value** of the action – typically damage or healing amount – along with damage type, critical hit flags, and mitigation info. This field is rich with information and its format can vary slightly depending on the event type:

- **Damage values:** A damage event's value field contains the amount of damage dealt, an optional second number (separated by a tilde `~`), the damage type, and possibly mitigation details:
- *Amount:* The primary number is the effective damage done to the target's health. For example, in `(40523 )` the number 40523 is the damage dealt.
- *Tilde (`~`) number:* If present, the second number after a `~` represents the **original roll or potential damage** before adjustments like overkill. In SWTOR, this is often used to indicate **overkill** damage or minor differences due to rounding. For instance, an entry `(594 ~596 energy {836045448940874})` means 594 damage was actually applied, whereas 596 would have been the full damage if the target had enough health (so 2 points were overkill) [11]. In other words, the target only had 594 HP left, so the remaining 2 damage was wasted. We often see very small differences (1–2 points) due to this overkill mechanic on killing blows. If the first number is smaller than the second, it indicates overkill. Conversely, if the first is larger (e.g. `937 ~936`), it may reflect rounding up of fractional damage (the game might calculate 936.4 and display 937, but logs also show the truncated/rounded counterpart). In practice, the tilde value can be ignored for total damage done (it's mostly for completeness); however, it's crucial for identifying overkill and understanding slight discrepancies between damage done and threat (since threat might be based on full damage).
- *Damage type:* After the numbers, the damage type is given as a keyword and an ID. For example, `energy {836045448940874}` or `internal {836045448940876}`. Common damage types

include **kinetic**, **energy**, **internal**, **elemental**, etc. Each has a unique ID (e.g. 836045448940874 for energy) [17] [25] . These indicate the nature of the damage and are important for understanding mitigation (armor mitigates kinetic/energy but not internal/elemental). If a shield absorbed part of the hit or the hit was deflected, additional text appears:

  - `-shield (X absorbed {ID})`: This indicates a portion of damage was absorbed by a shield. For example, `(25471 -shield {836045448945509} (15052 absorbed {836045448945511}))` means 25,471 damage was attempted, but a shielding effect absorbed 15,052 of it [26] . The remaining (25471 - 15052 = 10419) would be the actual HP loss. The IDs after `-shield` and after "absorbed" correspond to the game's internal effect IDs for a shield absorb and the absorb amount. A similar notation `-deflect` or `-miss` can appear if an attack misses or is deflected (though usually a miss is logged as an event without a value, since no damage occurs).
  - Multiple mitigation clauses can appear if relevant (e.g., `-shield` and also `-glance` if an attack glances, etc.), but typically you'll see at most one in a given event.

- *Critical hits:* A critical hit is denoted by a trailing asterisk `*` immediately after the first damage number. For example, `881* ~880` indicates the hit was a critical (the base damage was around 880, and 881 was the actual crit result after bonus) [27] . Non-critical hits have no asterisk. Heals can also crit, and similarly show an asterisk on the heal amount when they do.

- **Healing values:** Heal events use a similar `(amount ~overflow)` format with the same notation but interpreted for healing:

- The first number is the actual HP healed on the target, and the number after `~` is the **overheal** (the healing that went beyond the target's missing health and was wasted). If the second number is 0 or omitted, it means no overheal. For example, a heal might show `(7675 ~0)` meaning 7,675 health restored and none overhealed [28] . If a heal reads `(17150* ~0)`, it was a critical heal of 17,150 with no overheal [29] .
- If a heal does overheal, you would see something like `(10583 ~2116)` for instance, meaning 10,583 was actually applied and 2,116 was overheal (the ~ value for heals explicitly indicates overheal) [30] . In fact, in the logs we have, most heals are capped by targets' max HP, resulting in frequent `~0` second values. But whenever the target was full or nearly full, that second number would show the excess.

- Heals also include the type and ID of the heal effect. All direct healing uses the generic "Heal {836045448945500}" effect ID, and there is no damage-type per se (the game doesn't categorize heals like damage types). There is no shield notation for heals (shields only apply to damage).

- **Non-damage events:** Some events like applying a buff (ApplyEffect of a buff) or a pure event (AbilityActivate, TargetSet, etc.) do not have a meaningful numeric value. In those cases, the parentheses may be omitted entirely or contain a code. For example, in the `AreaEntered` event in the earlier example, the value field was `(he3001)` [31] . This appears to be a code rather than a numeric value – possibly an internal identifier for the location or event. Such codes are not documented, but they tend to appear in certain environment or phase transitions (e.g., `he3001` might stand for a specific environmental state). In most combat parsing, you won't need to interpret these codes; they can be safely ignored. Regular ability activations (`Event: AbilityActivate`)

also lack a numeric value – they simply mark that an ability was used, with no immediate effect on health or threat, so they appear without parentheses.

Threat (Aggro) Values

The last part of a log line, if present, is the threat delta enclosed in angle brackets `<...>`. This number indicates how much **threat** (aggro) was generated toward the target of the action. Threat is crucial for understanding enemy targeting in group content, and the log explicitly reports it for each damage/heal.

Examples from the log:
- A damage line `... (4582 energy {…}) <13747>` shows that 4,582 damage caused a 13,747 threat increase [17]. In this case, the threat is higher than damage because the source was likely a tank with threat modifiers (tank stances multiply threat – here ~3× damage). Indeed, 4,582 * 3 = ~13,746, matching the threat value. Non-tanks typically see threat equal to damage (1:1).
- A heal line `... (17150* ) <3430>` indicates a 17,150 heal generated 3,430 threat [18]. Healing threat in SWTOR is generally lower than damage threat. These logs suggest healing generates 20% threat of the effective heal (3,430 is 20% of 17,150). This aligns with known mechanics: healing produces less aggro than damage (with some nuances such as who the threat is attributed to – it usually splits among all enemies in combat).

Threat values allow parsers and tools to calculate hate lists and determine who has aggro. Note that threat is only logged for actions that actually modify threat (damage and heal primarily). Pure buff applications or non-combat events do not show a `<...>` threat number (or it may implicitly be zero). If an ability has threat-dropping or threat-transferring mechanics (e.g. a Guard ability or a detaunt), that can be inferred by unusual threat values (or separate log lines for those effects).

Edge case: Some lines might show a negative threat in angle brackets (e.g. `<-500>` if an ability explicitly lowers threat). However, in our experience, SWTOR logs threat drops as separate effects (like *ThreatDrop* abilities cause an ApplyEffect of a negative-threat buff or an Event). The `<...>` field itself usually represents positive threat gain. It's safe to interpret any `<number>` as the amount of threat added for that action on its target.

In summary, the angle-bracket threat number is one of the most useful features for advanced analysis (tank aggro management, etc.), and it became even more useful post-7.0 when we could see *everyone's* threat changes (since group members' actions are now included in the log). Community tools like StarParse display these threat values to help tanks monitor aggro [32].

Changes Introduced in Game Update 7.0

Game Update 7.0 (Feb 2022, *Legacy of the Sith*) brought **major changes** to SWTOR's combat logging system. These changes aimed to provide more detailed data (enabling modern parses and features akin to Warcraft Logs) and to reflect the new combat mechanics (like Combat Styles). Below are the key updates and their implications:

- **Logging of All Group Combat:** Prior to 7.0, your combat log only included events for **you and your companion** (anything you did, anything done to you, your companion's actions, and NPCs directly

interacting with you). Patch 7.0 removed this limitation: the log now records **all group members'** combat actions in an instance [4] [5]. This means if you are in an Operation or Flashpoint, your log will show ability uses, damage, and healing done by your teammates and to your teammates, not just yourself. In the example log, we see entries for other players like `@Harcanius`, `@Viszlo`, `@Only'flans`, etc., performing abilities and taking damage [33] [34]. The inclusion of group data was a huge change; it essentially allows a single log to capture the entire operation's events, which is why **parsers after 7.0 can compute everyone's DPS/HPS from one person's log**. (It also meant you "can't hide your performance" in group, as all damage/healing gets recorded and can be uploaded [4].) This change increased log file sizes substantially and made parsing more complex (needing to separate players by unique IDs, etc.), but it brought SWTOR in line with other MMOs that have full raid logging.

- **Coordinates and Orientation:** As noted earlier, 7.0 added the `(x,y,z,facing)` coordinates to every source and target entry [35]. These coordinates unlocked features like real-time mapping and fight replay visualizations. For instance, community developers quickly took advantage of this to create overlays: one Reddit user demonstrated a "second screen real-time map" plotting group members via log coordinates [35]. Coordinates were not present in pre-7.0 logs; now they are always included, even for events like buff applications or healing. The facing angle can be used (in theory) to analyze orientation-specific mechanics (e.g. whether a boss was facing the group or turned away at a given moment).

- **Health values for all entities:** Also new in 7.0, the `(current/max)` health is reported for not just the player but *every* source and target. In older logs, health might have been logged for the player in some context (and not at all for others). Now, you can see an NPC's max HP and track its health over time (as we did with Lord Kanoth's HP above). This is extremely useful for calculating **percent-based mechanics** and checking boss burn phases etc., and it allows parsers to display encounter timelines (e.g. graphing boss HP%). It also helps confirm when an enemy died (you'll see their HP drop to 0, often accompanied by a `Death` event).

- **Unique Identifiers for Players:** SWTOR always had numeric IDs for NPCs (the ones in `{braces}`), but players did not have a similar ID visible (since their names are unique per server, collisions are rare). In 7.0, players are now tagged with an explicit unique number after a `#`. For example, `@Menkey#686431891595421` and `@Menkey#123456...` would distinguish two players named Menkey (if that were possible) or more practically the same player across sessions. The exact nature of this ID isn't officially documented, but it may be an account or legacy identifier hashed for the session, or simply a unique session ID for that character. In any case, it ensures that if two players have the same name (which can happen now with cross-server group content or stronghold dummies named after players), the log lines can be attributed correctly. It's also useful for tracking one player across multiple log files or reconnects – the ID tends to persist through the session. **Note:** Companions and NPCs also have the two-part identifier with a colon as described, which was either introduced or standardized in 7.0.

- **Version Tags:** In some logs, especially the first line of a new log file, you might see a version tag like `<v7.0.0b>` or similar at the end of the line [31]. This appears in the `AreaEntered` event when logging starts. It denotes the game client version. In our example, `<v7.0.0b>` was likely the build version (7.0.0b). This is useful for parsers if any format changes by version; a parser could adjust logic based on the presence of a version tag. For instance, logs from 6.x won't have coordinates or

these tags, so seeing `<v7.0.x>` confirms the new format. After major patches, the letter may change (7.1.0, etc.). It's good practice to ignore or record this quietly, as it's not an actual combat event but metadata.

- **New Event Types:** With 7.0's overhaul, a few new kinds of events started appearing:

- `DisciplineChanged` events (as mentioned) when you swap loadouts or disciplines out of combat [24].
- `AreaEntered` / `AreaExited` or similar, logging the location changes (useful for multi-zone operations or tracking where an encounter took place).
- `ChallengeBegin` and `ChallengeEnd` for PvP matches or perhaps some Galactic Starfighter matches (though GSF has its own log format).
- **Loadouts** are not explicitly logged, but changing one triggers the discipline change event and reapplication of buffs. (In the logs, when Pugzeroone changed discipline, many buffs like Sprint were removed and reapplied automatically [36].)

- Cross-faction buff application: due to 7.0 allowing Republic and Imperial classes to play together, using a class buff applies *both factions'* versions of the buffs. In the log, this can appear as multiple ApplyEffect lines for each buff variant. In our example, the Trooper's **Fortification** buff caused not only the Republic buff effects (Force Might, Force Valor, Lucky Shots, Fortification) to apply to Pugzeroone, but also the Imperial versions (Coordination, Unnatural Might, Mark of Power, Hunter's Boon) to apply – the latter set was attributed to Pugzeroone's companion (Z0-0M) in the log [15] [37]. This was a quirk of how the game granted those cross-faction buffs (likely a minor bug or implementation detail), but it illustrates a 7.0 change: **all class buffs are granted at once** if you have the legacy unlocks, and both factions' buffs will appear in logs when in a mixed group.

- **Performance considerations:** With group logging on, the volume of log data skyrocketed. An 8-player operation log will include every ability from 8 players + companions + all NPCs. BioWare actually warned players that combat logging could generate very large files ("feature for advanced users that may require active disk management" as the UI says [38]). After 7.0, it's not uncommon to see log files tens of MB in size for a single operation night. This change meant **parsers had to be more efficient** and many were updated. For example, Parsely.io underwent a "big revamp" to handle the new data and allow detailed analysis and replays [9], and SWTOR's own community parsing programs (StarParse, etc.) had to update to filter or summarize other players' info [32].

In summary, patch 7.0 turned SWTOR's combat log from a personal diary into a group-combat black box recorder. The added data (positions, full group events, unique IDs) brought SWTOR's logs to a modern standard, at the cost of complexity and file size. Any reference or tool from before 2022 is now outdated in terms of log format – parsers had to adapt significantly.

Known Inconsistencies, Anomalies, and Edge Cases

While SWTOR's combat log is very comprehensive, it isn't without quirks. Here we catalog various oddities and edge cases that can confuse interpretation or parsing:

- **Simultaneous events and ordering:** The log is chronological, but events that occur in the same instant can appear in rapid sequence with the same timestamp (down to the millisecond). For

example, an ability that hits multiple targets (AoE) or a player using a channeled ability can produce several lines with identical timestamps. SWTOR does not guarantee any particular ordering for events within the same millisecond. AOE hits might be logged in target ID order or some internal order. As a parser, you should not assume that the log's line order strictly reflects causality if the timestamps are equal. Fortunately, true simultaneous events are rare and typically independent (e.g. multiple targets taking damage at once). Just be aware that sorting by timestamp alone might lump together actions that are effectively concurrent. In the example of an AoE (**Death From Above** in an Operation log), multiple enemies took damage at `01:31:33.000` exactly, each logged on separate lines [39].

- **Effect application vs. resolution delays:** Some abilities in SWTOR have a wind-up or travel time. The log might first show an `AbilityActivate` event for the cast start, and the actual `ApplyEffect: Damage` line appears a few moments later when the ability lands. This can be a "delay" in real terms but is expected (not really a log error). However, certain buff effects or procs can appear to happen *before* the ability that triggered them, due to how the game calculates events. For example, a buff like *Pulse Engine* (which procs off Hammer Shot) was logged at 01:28:40.284, *before* the Hammer Shot's own damage at 01:28:40.285 in one sequence [40] [7]. The game likely applied the buff instantly on activation, then dealt damage a millisecond later. This can seem out of order if you expected damage then buff. Understanding the ability mechanics (Pulse Engine gives a stacking buff on use of Hammer Shot) clarifies this ordering. When parsing, it might be necessary to group events by activation cycles rather than strict timestamp order in such cases.

- **Duplicate or redundant entries:** Certain state changes can cause rapid RemoveEffect/ApplyEffect pairs that look like duplicates. A common example is **Sprint**. When you change zones or disciplines, the game often removes the Sprint effect and then reapplies it immediately (perhaps to recalculate movement speed). In our log, at 01:14:42.356, upon switching discipline, we see Sprint being removed and deactivated, then at 01:14:42.356 again Sprint re-applied [36]. These show up as three lines all in the same millisecond: a RemoveEffect, an AbilityDeactivate (for the Sprint ability toggle off), and an ApplyEffect (re-applying Sprint). To the untrained eye it looks like spam or duplicated events, but it's actually the game toggling a persistent ability. Another scenario is class buffs when joining a group: if you log in or zone in, the game may automatically apply your buffs, and if they were already present it might remove and reapply them to sync durations. This can result in buff ApplyEffect lines that appear twice in a row for the same buff. Parsers might need to suppress immediate reapplications if they want to avoid double-counting buffs.

- **Delayed or uncaptured damage:** Most damage sources are logged, but a few environmental damages are tricky. **Falling damage** is one – if you jump off a high place and die, the log will show a `Death` event, but might not show a damage line for the fall (because fall damage is not attributed to an ability or source, it's environment). Some logs do record fall damage as an ApplyEffect by `<Environmental>` or similar with an ability "Fall" or an effect ID (this can vary). If you see a death with the target at 0 HP and no prior damage line, it could be an unreported environmental damage. Similarly, certain mechanic-based deaths (e.g. puzzle fails) might just log the death. In operations, usually every damage has a source (even if it's the encounter itself with an ability name). This is a minor edge case but worth noting: **not every HP loss has a clear damage line** (though this is rare).

- **Overkill and overheal rounding:** As discussed, the presence of those tilde values (~) can be confusing. A quirk: sometimes you'll see something like `(1 ~8423 …)`. A dramatic example: an

ability hits an enemy that only had 1 HP left, doing maybe 8,423 potential. The log will show it dealt 1 (killing blow) ~8423 (would have been) [41]. If you sum damage purely by the first number, you get the actual damage to the target's health (which is usually what you want for DPS calculations). Summing the second numbers would give you a higher "potential" damage sum which isn't useful except to identify wasted damage. For healers, summing the first number gives effective healing, summing the second gives raw healing if no overheal. Most parsers care about effective values. The anomaly is if you're unaware of this, you might wonder why a big attack appeared to hit for only "1". It did full damage, but only 1 was needed. When analyzing logs manually, keep an eye on the `~` in such cases.

- **Threat anomalies:** In some cases, threat numbers might not sum up straightforwardly. Abilities like Guard (which transfers threat from the guarded player to the tank) or taunts (which fixate threat) can make the threat field less intuitive. For instance, a Guarded DPS might deal 10,000 damage but only see <5000> threat in their log if 50% was redirected to the tank. Actually, the log likely still shows the full threat on the DPS's action, and separately the guard effect would log an aggro transfer. There is an effect called *Guard* that causes a `ApplyEffect: ModifyThreat` or similar event – these might appear as negative threat on the DPS or positive on the tank. This is advanced, but just be aware that threat can be directly modified by certain abilities. The log does capture these, often as their own lines (e.g., an ability with an effect "Threat Drop" will show a negative threat value in angle brackets). If you encounter an `<-xxxx>` threat, that's an intentional threat reduction ability at work.

- **Companions and mirror abilities:** Companions mimic player abilities in some cases and those appear in logs with companion as source. Also, since 7.0 introduced Combat Style swapping, you might see ability names that don't "belong" to a player's class if they swapped styles. E.g., an Agent ability used by a Smuggler (because that Smuggler took an Imperial Combat Style). The log will list the ability by its *actual* name, which might confuse if you assume faction. But because logs use the unique IDs, mirror abilities usually share the same ID across factions (e.g., *Collapse* vs *Back Blast* might have different names but perhaps share an underlying effect ID or at least are distinguishable by ID). An anomaly can arise if an ability's name is identical to another (rare in SWTOR) – always rely on the ID for 100% accuracy.

- **Missing buffs or stack updates:** Not every buff stack change is individually logged. Stacks of buffs or debuffs (like stacking DoTs) often update via reapplication events, but sometimes the log will show something like `ApplyEffect ... (5 charges)` indicating the new stack count [42]. In our example, `Power Screen {2842082938978304}` was applied with "(1 charges)" [42]. If a buff simply increments a stack, SWTOR might re-log the ApplyEffect with the new stack count or might only log the first apply and the final removal. This can be inconsistent per ability. Thus, if you're tracking buff stacks, you may need to infer stack increases either by repeated ApplyEffect lines or by counting occurrences. It's an inconsistency in how different effects are logged.

- **Combat start/end detection:** There's no explicit "combat started" line for players, but there is `Event: EnterCombat` for NPCs (especially bosses) which can hint at when a fight began. For players, you typically detect combat start by the first ability use or being hit, and combat end by a short silence followed by the `ExitCombat` or simply the absence of combat events for a period. SWTOR doesn't log an `ExitCombat` event for players consistently, though it does for things like PvP challenges. Some logs show `Event: ExitCombat` for the group at the end of a boss fight

(possibly introduced in 7.x). If it appears, use it; if not, the end of combat can be inferred by the `Death` of the boss or a gap.

- **Heals on dead targets / wasted heals:** If a heal lands right as a target dies or is at full health, you might see the heal value with full overheal. If the target is dead, the heal might not apply at all (and possibly won't log, or logs as full overheal on a 0 HP target – this is a corner case requiring careful reading).

In general, SWTOR's logs are very reliable. The "anomalies" mostly require understanding the game mechanics rather than the log misreporting. The important takeaway for parsers is to handle things like simultaneous timestamps, repeated apply/remove, and the presence of those tilde values properly.

Parsing Best Practices

Parsing SWTOR combat logs can be challenging due to their detail and volume, especially post-7.0. Here are some best practices and tips for processing these logs, whether you are writing a parser library or analyzing them manually:

- **Read line-by-line, sequentially:** The log is naturally ordered by time, so reading it sequentially is the simplest approach. Each line is independent in format, but context (previous lines) can matter for interpreting sequences (e.g. an ability activation followed by its effect). Tailing the log in real-time (for a live DPS meter) is usually done by reading new lines as they are written. If doing this, make sure to handle partial lines (the game writes line-by-line, so partial writes are uncommon except maybe on crash).

- **Use a reliable parsing method (regex or state machine):** The line format is consistent, so a well-crafted regular expression can capture all fields. For example, a regex (simplified) might break the line into groups: timestamp, source (with subgroups for name, IDs, coords, health), target, ability, event, effect, value (with possible subgroups), and threat. Many community parsers have published patterns. For instance, one could regex something like:

```
^\[(\d{2}:\d{2}:\d{2}\.\d{3})\] \[(.*?)\] \[(.*?)\] \[(.*?)\] \[(.*?):
(.*?)\](?: \((.*?)\))?(?: <(\-?\d+)>)?$
```

  (This is an oversimplified pattern for illustration – actual parsing needs to account for the nested braces and special characters in names.) If using regex, be mindful of characters like apostrophes in names (e.g., `Only'flans`) and spaces. Alternatively, split by `] [` boundaries, but remember the first token starts with `[` and others end with `]` etc. Many developers find it easier to parse by locating the consistent delimiters (`] [` and the known order of braces and parentheses) rather than one giant regex.

- **Threading and performance:** Given log files can be huge, you might be tempted to parse in parallel. You can split the file by chunks (e.g., by time or by lines) and have threads parse sections, but **be very careful to merge results in chronological order.** If you simply split the file without regard to event order, you could mis-order events that straddle a chunk boundary. A safer approach

is to use a single producer (reading the file sequentially) and multiple consumers (e.g., one thread handling damage calculations, one handling threat, etc.) if you want parallel processing of analysis, not of raw parsing. In practice, a well-written parser in a modern language can parse even very large logs in seconds, so multithreading the parsing itself is often unnecessary. The community-built parsers in C# and Python typically do a single-thread pass to convert the log to structured events, then allow multi-threaded analysis on the in-memory data if needed.

- **Memory considerations:** Streaming the log line by line (especially for real-time reading) is ideal to keep memory usage low. If you need random access (for example, to detect fight boundaries and then analyze within them), you might first parse into an array of event objects. For extremely large logs, consider indexing by timestamp or by encounter breaks (e.g., when a boss dies, mark that index) to avoid scanning everything repeatedly.

- **Maintain state for certain events:** Some calculations require keeping track of state as you parse:

- Tracking current buffs on entities (apply vs remove).
- Current targets of players (if analyzing target swapping).
- Accumulating damage/healing totals per combat segment.
- For threat, tracking each NPC's threat table if you want to know who has aggro – you'd add up threat per source, reset on drop events, etc.
- If you're reconstructing an encounter timeline, note the timestamp of first and last action, boss health at various points, etc.

Maintaining dictionaries or maps keyed by unique IDs (for players, NPCs) is helpful. E.g., a map of `entityID -> { currentHP, maxHP, buffs[] }` updated by each line. The unique instance IDs mean you can differentiate multiple mobs of the same name – consider using the full `Name{ID}:Instance` as a key for NPCs.

- **Be mindful of line breaks and encoding:** Logs are text files encoded in UTF-8. They may contain special characters (em dash, apostrophes, etc.) because player or NPC names can include them (as seen with "Too Swole-to'control" – note the hyphen and apostrophe). Ensure your parser handles UTF-8 and doesn't choke on those characters. Each log entry is terminated by a newline. Occasionally, a logging bug might produce a blank line or a line that starts or ends unexpectedly (these are rare). It's wise to have a fallback in the parser: if a line doesn't match the expected pattern, you can skip it or try to handle it gracefully rather than crash.

- **Correlating related events:** Often you'll want to link an `AbilityActivate` event with the subsequent `ApplyEffect` that resulted. The log itself doesn't explicitly tie them (no common ID besides timing and maybe ability name). A best-effort approach is:

- When you see an `AbilityActivate [Ability X]` for a source, assume the next ApplyEffect by that same source with that ability (or related effect) is the result. This works for instant and casted abilities. Channeled abilities might show multiple ApplyEffects after one activate.
- For damage over time (DoT) abilities, the initial cast is separate from the periodic damage ticks (which appear as their own ApplyEffect events on intervals). You might map them by ability name or effect ID. For example, *Affliction* cast by a Sorcerer: you'll get `AbilityActivate: Affliction`

and then periodic `ApplyEffect: Damage (internal)` events with effect "Affliction" on the target. The cast line and damage lines share the effect name "Affliction", so you can correlate them.

- Some high-level analyses might want to group all events of a single ability usage (e.g., a Marauder's Smash applies damage to 5 targets – 5 lines – plus one AbilityActivate line; grouping those 6 lines as one "ability usage" unit can be useful). This requires looking at source, ability, and a short time window. We mention this because if writing a parser, you might implement such grouping after the initial parsing, not during, to keep the first pass simple.

- **Multi-threaded environment caution:** If multiple processes or threads might read and write to the same log file (for instance, if you have the game writing to it and a parser reading it concurrently, or two parsers trying to read the same file), use file-locking or read snapshots. Typically, reading does not lock the file – you can tail it while the game writes. Just handle the end-of-file carefully (don't hold the file open in a mode that prevents writing). In a multi-threaded parser program, protect any shared data structures (like global counters) with locks or use thread-safe queues for communication. The log lines themselves should be processed in order; parallelizing within a single log's events can lead to out-of-order handling unless you take special care.

- **Separating fights or segments:** Often you'll want to break the continuous log into discrete encounters (boss fights, trash pulls, PvP matches). SWTOR doesn't label fights explicitly, but some markers help:

- Boss NPC `Death` events indicate an encounter end in operations.
- Long gaps (e.g., >30 seconds no combat) usually indicate downtime between fights.
- EnterCombat/ExitCombat events for bosses or players.

- You may also reset your parser's counters when the operation group composition changes or when an `AreaEntered` for a new map is seen (indicating moving to a new level or instance). Tools like StarParse allow users to manually mark segments or automatically break on boss deaths. Your approach can vary depending on needs (simple case: treat every boss death as end of a segment).

- **Validate with known totals:** A good practice after parsing is to validate some basic totals against the game's known outcomes. For example, sum of all damage dealt to a boss should equal the boss's max HP (plus any overkill) by the time it dies. Or, a simpler check: a player's total damage out minus damage absorbed should equal the sum of damage lines for that player as source. If numbers are off, you might be double-counting or missing lines. Community tools sometimes maintained cross-checks like these to ensure accuracy.

Lastly, leverage community resources: The SWTOR parsing community has produced open-source projects and documentation (see **References** below). For instance, *SWTORExtended* and *SWParse* projects on GitHub provide parsers you can study [43], and the **SWTOR Combat Parser Library** documentation (v5.0) contains detailed notes on handling various effects and edge cases. Studying those can give deeper insight into nuances like threat math and specific boss mechanic logging.

Community Tools and Corroboration

Over the years, several community tools have been built on top of SWTOR's combat logs. Using or referencing these tools can help validate your interpretations:

- **StarParse (Ixale's)** – A widely used real-time parser and overlay. StarParse was updated for 7.0 to handle the new log format (showing group DPS, for example) [32]. It is a closed-source tool, but it confirms things like: threat is being read from logs, damage/healing is summed from the log lines, etc. If StarParse shows a certain DPS or mechanic, you can usually trace it back in the raw log.

- **Parsely.io** – The long-running log upload site. Parsely underwent a major update post-7.0 to utilize coordinates and multiple players' data [9]. On Parsely, you can see encounter replays, damage distributions, and so on. This tool essentially parses the log file you upload and thus serves as a reference for how to correctly interpret lines. If your parser's output disagrees with Parsely on, say, total damage done by a player, that's a sign to revisit your handling of certain entries (accounting for overkill, etc.).

- **SWTOR Logs (swtorlogs.com)** – Another upload-and-analyze site, created by the same folks behind Warcraft Logs/FF Logs [44]. It supports advanced filtering and has a robust combat parser under the hood. SWTOR Logs took a few months to catch up to 7.0, but it's now a reliable reference. Their developer documentation (if any publicly available) might not be extensive, but they have forums where some mechanics are discussed. Using SWTOR Logs, you can inspect timelines of events, which implicitly confirms how logs are read (for example, you can see at what exact second each debuff was applied, meaning their parser picked up those lines correctly).

- **Open-source parsers:** There are open projects such as *SWTOR Combat Parser (SWCP)* and *SWParse*. The user provided a **SWTOR_Parser** documentation (v4/v5) which lists known issues and the interpretation of fields. For instance, it clarifies the use of the `*` for crits and `~` for overkill/overheal, aligning with what we've covered. If you prefer coding in Python, the **TeamSWAP/swap** project is an open-source parser in Python that you can refer to [45]. There's also a C# library *swtor-parser* by Ezet that might have a wiki or readme describing the format.

- **TORParse (historical)** – TORParse was an early web parser (no longer active) and several forum posts from 2012-2013 reference it. Those older discussions (e.g., on the SWTOR forums) helped establish the basic format [1] [2], which remains the foundation of today's logs. Just keep in mind any source from pre-2022 won't mention coordinates or group logging, but they are still useful for core concepts like the meaning of IDs and crit asterisks.

Using multiple tools on the same log can be instructive. If two different parsers agree on a value, it's likely correct. If you find a discrepancy (say one tool counts an extra tick of damage), it could highlight a tricky log aspect. The community has ironed out most parsing issues by cross-verification. Additionally, SWTOR developer posts are rare on this topic, but sometimes patch notes indirectly reveal things (e.g., "*Players may now choose to view victory and defeat messages in the chat window.*" indicates those events exist in logs [46]).

In conclusion, interpreting SWTOR combat logs is a combination of understanding the format and the game's mechanics. With the breakdown of fields above and careful parsing practices, one can extract a

trove of information – from DPS metrics to positional data and threat tables. The changes in 7.0 modernized the logs significantly, so any technical reference (like this document) should be considered up-to-date for 7.x and beyond. Future patches might extend logs further, but the current structure provides a robust framework for any combat analysis or tooling you wish to build.

References

- SWTOR Official Forums – **Combat Log Format Spec (2012)** – community discussion of log fields [1] [2]
- SWTOR Official Forums – **Combat Log Parser Round-Up & Instructions** – list of community parsers and basic log info [47]
- *torcommunity* – **SWTOR Combat Logging and Parsing Basics** – user guide summarizing log enabling and usage (archived)
- SWTOR Fan Blog *Going Commando* – **"Combat Logging in SWTOR" (Sept 11, 2022)** – overview of 7.0 logging changes and tools [4] [9]
- **Example SWTOR Combat Log Snippets** – Provided by user (combat logs from Sept 2023), demonstrating format with coords and group data [48] [49]
- Reddit – **/r/swtor discussion of 7.0 parsing ("Would you like a 2nd screen map…")** – confirming addition of coordinates in 7.0 logs [35]
- GitHub – **ezet/swtor-parser** – an open-source SWTOR log parsing library (C#) [43]
- GitHub – **TeamSWAP/swap (Star Wars Analyzer & Parser)** – open-source parser in Python [45]
- **SWTOR Parser DLL Documentation v5.0** – Technical documentation (user_files) on a community parser, including field specs and known issues (covers pre-7.0, but fundamental concepts still apply).
- Parsely.io – *SWTOR Combat Logs* online analyzer (Revamped for 7.0, allows group log upload and analysis)
- SWTORLogs.com – *SWTOR Combat Analysis by WarcraftLogs*, supports detailed queries (it's a live implementation of how to parse and use SWTOR logs data).

---

[1] [2] [3] Combat Log format spec - General Discussion - SWTOR | Forums
https://forums.swtor.com/topic/318158-combat-log-format-spec/

[4] [5] [9] [38] [44] Going Commando | A SWTOR Fan Blog: Combat Logging in SWTOR
https://swtorcommando.blogspot.com/2022/09/combat-logging-in-swtor.html

[6] [7] [8] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [33] [34] [36] [37] [39] [40] [41] [42] [48] [49] combat_2023-09-25_01_14_15_971249.txt
file://file-6R1qDTgFV2CLqrK5n6poQn

[32] SWTOR Logs - Combat Analysis for SWTOR
https://www.swtorlogs.com/

[35] Would you like a 2nd screen real-time map & "while you're ... - Reddit
https://www.reddit.com/r/swtor/comments/tigime/would_you_like_a_2nd_screen_realtime_map_while/

[43] ezet/swtor-parser: A library for parsing SWTOR combat logs - GitHub
https://github.com/ezet/swtor-parser

45 TeamSWAP/swap: SWTOR Analyzer and Parser - GitHub
https://github.com/TeamSWAP/swap

46 Legacy | Star Wars: The Old Republic
https://www.swtor.com/patchnotes/1.2.0/legacy

47 Combat Log Parser Round-Up & Instructions - SWTOR | Forums
https://forums.swtor.com/topic/347457-combat-log-parser-round-up-amp-instructions/