

KMProcsTracker — Developer Documentation (Retail)

Purpose

A minimal addon that tracks **Killing Machine** (spellID 51124) stacks on the player and counts:

- **Procs:** stack gains (0→1, 1→2)
- **Consumed:** stack losses (2→1, 1→0) — “stack loss”, not strictly “spent by Obliterate”
- **Wasted:** a proc event occurs while already at **2 stacks** (overcap), surfaced as a short UI warning

UI is intentionally minimal: a single icon with stack count, drag to move, resize grabber, and a right-click context menu.

File/Module Layout

KMProcsTracker.toc

Defines addon metadata, SavedVariables, and file load order.

Key points:

- `## SavedVariables: KMProcsTrackerDB`
- Load order must ensure `Core.lua` defines `Addon` table before `UI.lua` / `Minimap.lua` use it.

Core.lua

Owns:

- SavedVariables initialization and defaults
- Event wiring (combat start/stop, aura changes, combat log)
- All counters + state
- Public methods used by UI/minimap

UI.lua

Owns:

- Icon-only UI frame creation and behavior
- Stack display, visuals, “Wasted” overlay
- Drag/move + resize behavior

- Right-click context menu
- Combat-state alpha without modifying Core (uses `UnitAffectingCombat` + its own combat event listener)

Minimap.lua (optional)

LDB launcher + LibDBIcon integration; left click toggles visibility, right click resets.

Data Model

SavedVariables: `KMProcsTrackerDB`

Structure:

```
KMProcsTrackerDB = {
  minimap = { hide=false, minimapPos=220 },
  frame = {
    point="CENTER", relativePoint="CENTER", x=0, y=0,
    scale=1.0,
    shown=true,
    locked=false,
  }
}
```

Runtime State (non-saved)

In `Core.lua`:

```
Addon.state = {
  playerGUID = UnitGUID("player"),
  inCombat = false,
  manualPaused = false,
  stacks = 0,
  lastWastedAt = 0,
}

Addon.counters = {
  procs = 0,
  consumed = 0,
  wasted = 0,
}
```

Event Flow and Logic

1) Initialization

- `ADDON_LOADED:`
 - `Create KMProcsTrackerDB` if missing
 - `Apply default values recursively`
- `PLAYER_LOGIN:`
 - `Store playerGUID`
 - `Initialize UI/minimap`
 - `Sync current KM stacks once (out of combat; does not count)`

2) Combat gating (when counting is enabled)

Counting is enabled only when:

```
Addon:IsTrackingEnabled() == (state.inCombat == true and state.manualPaused == false)
```

Combat start/stop:

- `PLAYER_REGEN_DISABLED:`
 - `SyncStacksFromAura()` first (baseline; avoids counting pre-pull stacks as procs)
 - `state.inCombat = true`
- `PLAYER_REGEN_ENABLED:`
 - `state.inCombat = false`

3) Stack tracking (authoritative source)

Stacks are read via aura scan on `UNIT_AURA` for the player.

Aura scan

`Addon:GetKMStacksFromAuras()` uses:

- `AuraUtil.ForEachAura()` when available (Retail)
- fallback to `UnitAura()` loop if needed

Delta-based counting

`Addon:SyncStacksFromAura()`:

- reads `old = state.stacks, new = current aura stacks`
- updates `state.stacks = new`
- if tracking enabled:
 - `delta > 0 → counters.procs += delta`
 - `delta < 0 → counters.consumed += -delta`
- updates UI stacks display each time

This means “Consumed” is **stack loss**. If you want “spent only”, add a cast correlation layer (see Extensions).

4) Wasted detection (overcap at 2)

Wasted is counted from `COMBAT_LOG_EVENT_UNFILTERED`, filtered to:

- `destGUID == playerGUID`
- `spellId == 51124`
- `sourceGUID == playerGUID` (defensive filter)

Then:

- `only subevent == "SPELL_AURA_REFRESH"` is used as wasted signal
- wasted triggers if stacks were already at cap:
 - `state.stacks >= 2`
 - and if `amount` exists, require it `>= 2` (aura stays at 2)

Anti-double-count:

- uses timestamp delta threshold (`lastWastedAt`) to ignore duplicates within ~0.05s.

UI reaction:

- `UI:ShowWasted()` shows “Wasted” overlay and temporarily desaturates the icon.

UI Implementation Notes (UI.1ua)

Frame composition

- `Button` frame (not `Frame`) is used to reliably receive right-click and drag:
 - `RegisterForClicks("AnyUp")`
 - `RegisterForDrag("LeftButton")`

Children:

- `Texture` (icon)
- `Texture` (2-stack border glow)
- `FontString` (stack count, bottom-left)
- `FontString` (“Wasted” overlay)

Alpha rules

UI chooses alpha based on:

- combat state via `UnitAffectingCombat("player")`
- stacks state (0 stacks slightly dimmer)

This is intentionally independent of Core so no Core edits are required for out-of-combat dimming.

Drag/move

- `OnDragStart: StartMoving()` unless locked or resizing
- `OnDragStop: StopMovingOrSizing()` + save position to DB

Resize behavior (no drifting)

Resizing is done via `SetSize()`, not `SetScale()`:

- frame scale is forced to 1.0
- desired scale is stored as `db.frame.scale`
- actual pixel size is computed:
 - `size = BASE_ICON_SIZE * scale`
- border size and text scale are adjusted accordingly

Anchor strategy:

- when resizing starts, frame is temporarily re-anchored to **TOPLEFT** at its current screen coordinates:
 - `pinLeft = frame:GetLeft()`
 - `pinTop = frame:GetTop()`
 - `frame:SetPoint("TOPLEFT", UIParent, "BOTTOMLEFT", pinLeft, pinTop)`
- `SetSize()` preserves the pinned corner.

Grabber:

- remains at **BOTTOMRIGHT** as requested
- cursor delta uses `UIParent:GetEffectiveScale()` to keep cursor math stable.

Context menu

Right-click shows a context menu:

- preferred Retail: `MenuUtil.CreateContextMenu` if available
- fallback: `EasyMenu + UIDropDownMenuTemplate`

Menu actions call Core:

- Start/Stop toggles `manualPaused`

- `Reset` calls `Addon:ResetCounters()`
 - `Close` hides the frame (`db.frame.shown = false`)
-

Key Public Functions (API surface)

Core:

- `Addon:GetDB()`
- `Addon:IsTrackingEnabled()`
- `Addon:SetManualPaused(paused)`
- `Addon:ResetCounters()`
- `Addon:GetKMStacksFromAuras()`
- `Addon:SyncStacksFromAura()`

UI:

- `UI:OnLogin()`
- `UI:UpdateStacks(stacks)`
- `UI:ShowWasted()`
- `UI:ToggleShown()` / `UI:SetShown(bool)`

Minimap:

- `Minimap:OnLogin()`
-

Extension Points / Common Requests

A) “Consumed” as “spent-by-ability only”

Current consumed is stack loss. To count “spent”:

- Track casts that can consume KM (e.g., `Obliterate`, `Frostscythe`) via CL:
 - `SPELL_CAST_SUCCESS` from player
- When stacks drop on `UNIT_AURA`, attribute the drop to the most recent eligible cast within a short window (e.g., 0.2s).
This requires a small ring buffer of recent casts.

B) Make Wasted stricter

If some clients emit `SPELL_AURA_REFRESH` not only on overcap, tighten:

- require `state.stacks == 2` and `amount == 2` (when amount is present)
- optionally correlate with a detected proc source event (weapon swing/ability) if needed.

C) Slash config / GUI

Add a `/kmpt scale 1.2` or `/kmpt lock` command by extending `SlashCmdList`.

Debugging Tips

- Add a debug flag and print CL subevents for spellId 51124:
 - `subevent`, `amount`, `state.stacks`, `timestamps`
 - When debugging Wasted, log both:
 - the CL event and current aura stacks
 - the next `UNIT_AURA` update timing (order matters)
-

Design Constraints (intentional)

- Minimal UI footprint: no panels, no frames, no persistent counters on screen.
- Counters are maintained in Core and surfaced via tooltip/logic.
- No external dependencies required for core tracking; minimap is optional.

This document should be kept alongside the repo (e.g., `README_DEV.md`) and updated when logic changes (especially wasted detection and consumption rules).