ChatGPT

Updating the Outline Progress States and Schema

We will introduce two new pipeline phases: **SubPlotsDefined** and **BeatsExpanded**. These will be added to the `OutlineProgress` enum in the logical order of execution. Currently, `OutlineProgress` stops at `StructureOutlined` (with `BeatsDetailed` and `Finalized` reserved but not used) [1]. We will insert `SubPlotsDefined` right after `CharactersOutlined` and `BeatsExpanded` after `SubPlotsDefined`. This maintains the intended sequence for comparison helpers (e.g. `IsCharactersOutlinedOrLater()` checks `>= CharactersOutlined` [2]). The updated enum will look like:

```
public enum OutlineProgress {
    Init,
    PremiseExpanded,
    ArcDefined,
    CharactersOutlined,
    SubPlotsDefined,      // new phase after CharactersOutlined
    BeatsExpanded,        // new phase after SubPlotsDefined
    StructureOutlined,
    BeatsDetailed,
    Finalized
}
```

We will also update the JSON schema (`tools/outline.schema.v1.json`) to include these new phase names in the allowed `outlineProgress` enum [3]. Additionally, we'll add a new top-level field `"acts"` to the schema to capture act-level details. Each **act** object will contain:
- `number` (integer, act index),
- `definition` (string summary of the act),
- `beats` (array of strings for the act's major beats),
- `subPlots` (array of strings for subplot descriptions in that act).

This structure aligns with the requirement that *"Each act has: act number or label, definition, beats[], subPlots[]"*. We will define an `$defs.act` schema similar to `$defs.chapter` [4] [5]. The `acts` array will not be required at the `Init` stage, but once we populate it (at SubPlotsDefined stage), it must conform to the new schema definitions. We'll ensure `acts[].beats` can be an empty array or have a few short entries (max ~120 chars each, like chapter beats) and `acts[].subPlots` is a string array.

Implementing SubPlotDefinerService

This new service runs after the story arc is defined. In the pipeline, we will execute it once `outlineProgress == ArcDefined` or later. However, since the character roster is typically generated right after the arc, we plan to run SubPlotDefiner *after* CharactersOutlined to leverage any additional context if needed. (We will adjust the preconditions accordingly, see below.)

Outline Parsing: First, the service will parse the multi-act story arc text (`outline["storyArc"]`) into distinct acts. The arc text is stored as plain text paragraphs (one per act) with labels like "Act 1: … Act 2: …" separated by blank lines [6]. We can split this string on the "Act " labels or double newlines to isolate each act's summary. For example, the sample `storyArc` shows five act paragraphs labeled *Act 1* through *Act 5* [6]. We will strip out the "Act X:" prefix and store each act's content. Each act will be saved as an object in a new `outline["acts"]` JArray with fields: - `"number": X` (act index starting at 1), - `"definition": "<act summary text>"`, - `"beats": []`, - `"subPlots": []`.

We increment the outline's `schemaVersion` and set `outlineProgress` to **SubPlotsDefined** after this step.

Generating Subplot Threads: For each act, we call the LLM to generate **5–6 subplot ideas** relevant to that act. The prompt will use the **story genre, premise, target audience, and author preset** as inputs, per requirements. We'll build a prompt similar to the chapter subplot prompt in StructureOutliner, but at act-level. For reference, the existing StructureOutlinerService uses a prompt template for chapter subplots [7]. We will adapt this:

- Indicate the task and context, e.g. *"You are defining subplot threads to enrich Act X of the story."*
- Include key context: **Genre** and a brief **Premise** or act summary. We have the full premise (expanded to ~200–300 words in PremiseExpanded phase) and the act's definition. We can supply the premise for overall background, plus the act's specific situation to focus the subplots on that segment of the story.
- Include target **Audience** if `includeAudience` is true (similar to ArcDefiner including an audience line [8]).
- Include **Author style** if an author preset was requested. We'll use the famousAuthorPreset from the project file when `includeAuthorPreset` is true (just as CharactersOutliner does [9]), adding a line like *"Emulate the tonal style of <AuthorName>."*.

The prompt will then instruct the model to return **only a JSON array** of 5–6 concise subplot descriptions. For example:

```
"You are adding subplot threads to enrich Act 2 of the story.
Genre: Horror. Target audience: Adults (Ages 19–64).
Act 2 context: As the fog grows denser, villagers face nightmares and
apparitions... (brief act summary).
Return a JSON array with 5-6 concise subplot ideas."
```

We will use a robust parsing approach similar to StructureOutliner's `CallWithRetryAsync` and JSON sanitization [10] [11] to handle the LLM response. The resulting JArray (each element likely a string or short text) will be assigned to the act's `"subPlots"` field. We repeat this for each act in sequence.

Act-Level Beats (Optional): The requirement lists `beats[]` for each act. We will populate this to ensure a complete structure. A simple approach is to derive a few key beats from each act's definition. For now, we might split the act summary into a list of its major events or sentences (ensuring each is under ~120 chars to meet beat length guidelines [12] [13]). This provides a rough breakdown of the act. For example, if Act 1's definition mentions "the fog envelops the village… Old Man Cormac recounts tales… villagers live in uneasy

acceptance… the fog signals an impending threat," we can list those as beats. These can be generated either by simple sentence parsing or an LLM prompt if needed. We will then have an array of beats (3–6) for each act, stored in `act["beats"]`.

Finally, we update `outline["outlineProgress"] = "SubPlotsDefined"` and bump the `schemaVersion`. The resulting outline at this stage will have a new top-level `"acts"` array with each act's number, definition, beats, and subPlots populated.

Implementing BeatExpanderService

This service runs after subplots are defined (outlineProgress == SubPlotsDefined). Its job is to **expand the story beats and map them onto chapters**. By the end, `outline["chapters"][].beats` will be filled with detailed beats for every chapter. We set `outlineProgress` to **BeatsExpanded** after completion. Key steps:

Preparing Chapter Placeholders: Before generating beats, we ensure the `chapters` array has entries for all chapters (and any prologue/epilogue). Initially, the outline's `chapters` was an empty array [14] and hasn't been populated yet (since we skipped the old StructureOutliner step). We will create placeholder chapter objects now: for each chapter number 1 through `chapterCount`, add an object `{ "number": n }`. If the project has `includePrologue` or `includeEpilogue` set (as seen in the project JSON [15]), we also add a chapter `{ "number": 0 }` for the prologue and `{ "number": chapterCount+1 }` for the epilogue. This matches how StructureOutliner expected chapter entries for 0 and N+1 if those were to be included [16] [17]. These placeholder objects will allow us to insert beats into the correct chapters after generation.

Determining Act Count: We will use the specified act-count rule to guide the chapter distribution: - 3 acts if `chapterCount ≤ 15`
- 4 acts if `16 ≤ chapterCount ≤ 24`
- 5 acts if `chapterCount ≥ 25`

We'll compute the desired number of acts (`desiredActCount`). If the number of acts in our outline (from the `acts` array) does not match this, we may adjust. For instance, if the LLM gave 5 acts for a 20-chapter story (as in our example, where 20 chapters would ideally mean 4 acts by the rule), we will combine or trim acts to honor the rule. We prefer not to lose content, so merging the last two acts is an approach: e.g. merge Act 4 and Act 5 definitions and subplots into a single Act 4, thereby reducing the count. This preserves all subplot threads but now under the adjusted act. Conversely, if the outline had fewer acts than the rule, we may accept it (since generating entirely new act content is not feasible without another LLM call). The rule's primary purpose is to guide how we allocate chapters per act for beat distribution.

LLM Prompt for Expanded Beats: With acts, subplots, and characters available, we will craft a prompt to generate the chapter-by-chapter beats. This prompt will provide the model with all necessary context: the story's structure by acts (including subplots) and the character roster. We want the model to integrate subplot threads and character involvement into the beats of each chapter. The prompt will be along these lines:

- State the task, e.g. *"You are expanding the outline into detailed chapters with beats."*

- Provide high-level parameters: **Genre** and number of **Chapters**. For example, *"Genre: Horror"* and *"Core chapters: 20"* (mirroring the format used in StructureOutliner [18]). We also include the audience if needed [19]. If a prologue/epilogue is present, instruct the model to include those (e.g. *"Include a Prologue (number 0). Include an Epilogue (number N+1)."* just as the structure prompt did [16] [17]).
- **Story Acts and Subplots**: We list each act's summary and its subplot threads. For clarity, we can format this as:

```
Act 1: <Act 1 definition>
Subplots:
- <subplot 1>
- <subplot 2>
... (for all subplots of Act 1)

Act 2: <Act 2 definition>
Subplots:
- <subplot 1>
... etc.
```

  This gives the model a clear breakdown of the main storyline per act and the subplot threads to incorporate for each act. (We include the act definitions because the StructureOutliner's prompt did not include the story specifics, which led to the model somehow using them implicitly. By providing acts and subplots explicitly, we ensure the model has the necessary story context. In our example, names like Eleanor and Cormac appeared in chapter outlines, even though the original structure prompt didn't mention them – implying some missing context. We will explicitly include them now.)
- **Characters**: We add a summary of key characters so the model can involve them appropriately. A concise way is to list them by role and name. For example: *"Characters: Protagonist – Eleanor Thorne; Supporting – Old Man Cormac, Thomas Whitaker; Minor – Martha Finn, Samuel Lark."* This ensures the model knows which characters exist and can weave them into events (the model will recognize names from the context). We don't include their full arcs to save prompt space, but we do provide the roster and roles.
- **Instructions for Output**: We ask for a JSON array of chapter beats. Specifically, we say something like: *"Now outline the story in [chapterCount] chapters (plus prologue/epilogue if applicable). For each chapter, list its key beats (events) – 3 to 6 beats per chapter – incorporating the main plot and subplots with the characters. Return a JSON array where each item has:* `number` *and* `beats` *(the list of beats)."* We do **not** ask for titles, summaries or themes this time, only the beats array for each chapter, since our focus is on expanded beats. This final instruction ensures the LLM returns structured JSON we can parse easily [20].

Using the above prompt structure, the LLM (e.g. GPT-4) will generate a JSON array of chapters. Each element will look roughly like:

```
{ "number": 1, "beats": [ "First event ...", "Next event ...", ... ] }
```

including chapter 0 (Prologue) and chapter N+1 (Epilogue) if those were requested.

Parsing and Saving Beats: We will parse the LLM response as JSON (using the same `CallWithRetryAsync` mechanism to ensure valid JSON [10] [21]). Then we iterate through the resulting array and match each item to the corresponding chapter object in `outline["chapters"]` by number, just like StructureOutliner does for chapters [22]. We set each chapter's `"beats"` field to the generated beats array. (If any chapter was returned with a title or summary, we could incorporate it, but since we only asked for beats, we'll ignore extra fields or avoid requesting them at all.) We also leave chapter `"subPlots"` arrays as empty or omit them, since the subplot content is now integrated into the beats themselves.

Finally, we update `outlineProgress` to **BeatsExpanded** and increment the schema version. The outline now has a detailed beats breakdown for every chapter. (Any previously empty fields like chapter titles or summaries remain empty; we could consider generating chapter titles/summaries, but the requirement did not request it, and those can be added later or derived from beats if needed.)

CLI Integration (Commands and Pipeline)

We will add new CLI verbs for these stages in **Novelist.Cli/Program.cs**. In the command dispatch switch [23], we insert:

```
("outline", "subplots") => RunSubplots(args[2..], client, authorPreset,
includeAud),
("outline", "beats")    => RunBeats(args[2..], client, authorPreset,
includeAud),
```

We will implement `RunSubplots` and `RunBeats` similar to existing ones (Arc, Characters, Structure):

```
private static int RunSubplots(string[] a, ILlmClient llm, bool preset, bool
aud) {
    if (!Try(a, "--outline", out var o)) return Err("--outline required");
    var model = Get(a, "--model", "gpt-4o-mini");
    new SubPlotDefinerService(llm, preset, aud).DefineSubplotsAsync(o,
model).Wait();
    SaveSnapshot(o);
    return Ok("Subplots defined.");
}
private static int RunBeats(string[] a, ILlmClient llm, bool preset, bool aud) {
    if (!Try(a, "--outline", out var o)) return Err("--outline required");
    var model = Get(a, "--model", "gpt-4o-mini");
    new BeatExpanderService(llm, preset, aud).ExpandBeatsAsync(o, model).Wait();
    SaveSnapshot(o);
    return Ok("Beats expanded.");
}
```

We pass along the `--author-preset` flag (`preset` bool) and audience flag to these services. The services' constructors will accept those flags (similar to CharactersOutlinerService which uses `includeAuthorPreset` and Arc/Structure which use `includeAudience` flags). The `DefineSubplotsAsync` method will throw if the outline is not in the correct prior phase. We will set its precondition to require either `ArcDefined` or `CharactersOutlined`. Since our pipeline runs it after characters, we can check for `CharactersOutlined` to be safe (and similarly allow it if someone runs it immediately after ArcDefined, though then characters would come later). BeatExpanderService will verify that `outlineProgress == SubPlotsDefined` (and throw otherwise). Each service, upon success, updates the progress and writes the outline JSON to disk (and `SaveSnapshot` will create a timestamped snapshot with the new phase name, e.g. `outline_..._SubPlotsDefined.json`, `..._BeatsExpanded.json`).

We will also update the **run_novelist_live.bat** script to incorporate these new steps. Currently, after defining characters (Step 4), the script calls `define-structure` (Step 5) [24]. We will replace that:

- **Step 5: Define subplots** – run `novelist outline subplots --outline "%OUTLINE%" --model %MODEL% --live -s` (mirroring the flag usage).
- **Step 6: Expand beats** – run `novelist outline beats --outline "%OUTLINE%" --model %MODEL% --live -s`.

We will remove or comment out the old `define-structure` call, because the new two-step process replaces it. The arc and character steps (Steps 3 and 4) remain the same. After Step 6, we will echo that all steps are complete. The final `outlineProgress` will be "BeatsExpanded", and the final outline JSON will contain the premise, storyArc, characters, acts (with subplots), and fully expanded chapter beats.

Validation and Documentation

With the schema updated for the new fields and phases, the outline JSON should validate correctly after each stage. The snapshot mechanism will append the phase name to each saved file name, which helps us verify intermediate outputs (e.g. after running SubPlotDefinerService, `outlineProgress` is "SubPlotsDefined" and a snapshot `*_SubPlotsDefined.json` is saved, containing the new `acts` array with subplots). We should double-check that all required fields are present at each stage. For instance, after SubPlotsDefined, `acts` exists but `chapters` can remain empty until the beats stage; this is acceptable since `chapters` is required by schema but was already present (even if empty) from outline creation [25]. By the end of BeatsExpanded, every required field (including `chapters` with their beats) will be populated, satisfying the schema.

Finally, we will add documentation for these new stages in **requirements_digest.md** (or the relevant documentation file). We'll describe the purpose of the *SubPlotsDefined* and *BeatsExpanded* phases, the new CLI commands (`novelist outline subplots`, `novelist outline beats`), and how they fit into the pipeline after the Arc and Character definition stages. This documentation will also note the act-count rule and how chapters are divided among acts.

In summary, the new implementation will extend the outline-building process with a dedicated subplot definition stage (using the story arc and other inputs to enrich each act with subplot threads) and a beat

expansion stage (using the arc, subplots, and characters to produce detailed chapter beats). These changes align with the requirements and replace the older “structure” stage with a more narrative-aware approach, while ensuring the output JSON remains well-structured and valid.

Sources:

- Outline progress states in code [1] and schema [3].
- ArcDefinerService setting storyArc and ArcDefined [26].
- Example storyArc with acts labeled [6].
- StructureOutliner subplot prompt example [7].
- CharactersOutliner using author preset for style [9].
- StructureOutliner structure prompt (chapters and prologue/epilogue instructions) [27].
- Project JSON flags (prologue/epilogue, author preset) [15].
- Characters sample in outline (for roster context) [28] [29].

---

[1] OutlineProgress.cs
https://github.com/bilols/thenovelist/blob/3e7a893f93e2cc7ee7cfa1958fe045abdc55414b/src/Novelist.OutlineBuilder/OutlineProgress.cs

[2] OutlineProgressExtensions.cs
https://github.com/bilols/thenovelist/blob/3e7a893f93e2cc7ee7cfa1958fe045abdc55414b/src/Novelist.OutlineBuilder/Models/OutlineProgressExtensions.cs

[3] [4] [5] [12] [13] outline.schema.v1.json
https://github.com/bilols/thenovelist/blob/3e7a893f93e2cc7ee7cfa1958fe045abdc55414b/tools/outline.schema.v1.json

[6] outline_20250703212612.json
https://github.com/bilols/thenovelist/blob/3e7a893f93e2cc7ee7cfa1958fe045abdc55414b/outlines/outline_20250703212612.json

[7] [10] [11] [16] [17] [18] [19] [20] [21] [22] [27] StructureOutlinerService.cs
https://github.com/bilols/thenovelist/blob/3e7a893f93e2cc7ee7cfa1958fe045abdc55414b/src/Novelist.OutlineBuilder/StructureOutlinerService.cs

[8] [26] ArcDefinerService.cs
https://github.com/bilols/thenovelist/blob/3e7a893f93e2cc7ee7cfa1958fe045abdc55414b/src/Novelist.OutlineBuilder/ArcDefinerService.cs

[9] CharactersOutlinerService.cs
https://github.com/bilols/thenovelist/blob/3e7a893f93e2cc7ee7cfa1958fe045abdc55414b/src/Novelist.OutlineBuilder/CharactersOutlinerService.cs

[14] [25] OutlineBuilderService.cs
https://github.com/bilols/thenovelist/blob/3e7a893f93e2cc7ee7cfa1958fe045abdc55414b/src/Novelist.OutlineBuilder/OutlineBuilderService.cs

[15] thedoor.project.json
file://file-CzTDXM6ie77YqVWw4m4GRC

[23] Program.cs
https://github.com/bilols/thenovelist/blob/3e7a893f93e2cc7ee7cfa1958fe045abdc55414b/src/Novelist.Cli/Program.cs

24 run_novelist_live.bat
https://github.com/bilols/thenovelist/blob/3e7a893f93e2cc7ee7cfa1958fe045abdc55414b/run_novelist_live.bat

28 29 outline_20250703212612.json
file://file-Ec7nS3pFbRGcnpWA24RpJf