ChatGPT

Implementing New Outline Phases and Removing Structure Outliner

1. Adding New Outline Pipeline Phases

We introduce two new phases in the outline-building pipeline: **SubPlotsDefined** and **BeatsExpanded**. These will come **after** the CharactersOutlined phase and replace the old StructureOutlined phase. In the `OutlineProgress` enum (in *OutlineProgress.cs*), we add the new values and remove `StructureOutlined` [1]. The updated order becomes:

```
public enum OutlineProgress
{
    Init,
    PremiseExpanded,
    ArcDefined,
    CharactersOutlined,
    SubPlotsDefined,
    BeatsExpanded,
    BeatsDetailed,
    Finalized
}
```

This ensures the pipeline can progress through the new stages in order. We also update any helper methods that relied on `StructureOutlined`. For example, the extension method `IsStructureOutlinedOrLater` (in *OutlineProgressExtensions.cs*) should be adjusted to use the new phase – essentially checking for **BeatsExpanded or beyond** instead of StructureOutlined [2].

Correspondingly, we update the JSON schema **outline.schema.v1.json** to allow the new phase values. In the `outlineProgress` enum definition, we remove `"StructureOutlined"` and add `"SubPlotsDefined"` and `"BeatsExpanded"` [3]. After this change, an outline file's `"outlineProgress"` field can be one of: Init, PremiseExpanded, ArcDefined, CharactersOutlined, SubPlotsDefined, or BeatsExpanded (and we could include BeatsDetailed/Finalized if needed in future). This schema update reflects the new pipeline and prevents validation errors when the outline is in the new phases.

2. Removing the Old StructureOutlinerService

The **StructureOutlinerService** is now obsolete and will be removed entirely. This involves deleting the `StructureOutlinerService.cs` file and eliminating all references to it:

- **CLI command**: In *Program.cs*, remove the case for `"outline define-structure"` and its handler. Originally, the CLI parsed `"outline define-structure"` and called `RunStruct` [4]. We delete this case and the associated `RunStruct` method (which called `StructureOutlinerService.DefineStructureAsync` [5]).

- **Batch script**: In *run_novelist_live.bat*, remove the step that invokes the structure phase. For example, the original Step 5 was:

  ```
  echo STEP 5: Define structure
  dotnet run --project "%CLI%" -- outline define-structure --outline
  "%OUTLINE%" --model %MODEL% --live -s [6]
  ```

  We eliminate this so the batch no longer calls the structure outliner.

- **OutlineProgress**: As noted, we removed `StructureOutlined` from the enum and schema. Any logic checking for StructureOutlined (e.g. in extension methods or elsewhere) is updated to use the new phases as appropriate.

After these removals, the structure phase will not be invoked, and the new phases (SubPlots and Beats) will fill that role.

3. Implementing SubPlotDefinerService (SubPlotsDefined Phase)

We implement a new service **SubPlotDefinerService** to handle the SubPlotsDefined phase. Its responsibility is to **generate the story acts and subplot threads for each act**. Key points for implementation:

- **Preconditions**: It should only run on an outline that has completed CharactersOutlined. We enforce this by checking `outline["outlineProgress"] == "CharactersOutlined"` at the start, similar to how StructureOutlinerService checked for CharactersOutlined [7].

- **Load outline and project**: The service will load the outline JSON and also the original project JSON to get any needed settings. We find the project file via the outline header (the outline header stores `"projectFile"` name and the OutlineBuilder initially copied the project into the outline directory [8]). We then parse the project JSON. This allows us to read flags like `includePrologue` / `includeEpilogue` and other context. (The CharactersOutlinerService uses the same approach to retrieve the project file path [9].)

- **Initialize chapters structure**: Using the project info, we create chapter placeholders in the outline if they aren't already present. Specifically, if the project's `includePrologue` is true, we add a chapter object with `"number": 0` to represent the prologue. We then add chapter objects numbered 1 through `chapterCount` (from the project/outline) for the main chapters, and if `includeEpilogue` is true, add one with number `chapterCount+1` for the epilogue. At this point, these chapter objects will mostly be empty placeholders (just with number, possibly to be filled later). This step ensures that chapters for prologue/epilogue are not lost – they will be populated in the beats phase. It mirrors how the sample outline contains chapter 0 and 21 when prologue/epilogue are enabled [10] [11].

- **Determine acts**: We utilize the existing `storyArc` in the outline (generated in the ArcDefined phase) to identify the acts. The storyArc is a multi-paragraph string with each paragraph typically starting with "Act 1:", "Act 2:", etc. [12]. We parse this string to split it by acts. For each act, we capture its summary. We then create a new JSON structure in the outline, for example an `"acts"` array, where each item might include the act number and summary. We may also store subplots per act here (described next). *Note:* The outline schema v1 did not previously define an `"acts"` field – we can add it (as an optional field) to hold this structured act breakdown. Each act object can have a `number`, a `summary` (the act synopsis from storyArc), and a `subPlots` array.

- **Generate subplot threads per act**: For each act, we call the LLM to brainstorm 1-2 subplot ideas **for that act**. We build a prompt similar to the one used previously for chapter subplots, but at the act level. For example:

```
"You are outlining subplot threads for a novel act.\nGenre: <Genre>\nAct
<N> summary: <act summary from storyArc>...\nReturn a JSON array with 1-2
concise subplot ideas for this act."
```

  We include the story genre and (if `includeAudience` is true) the audience to guide tone, similar to other prompts [13]. We feed in the act's summary to ground the LLM in that portion of the story. The LLM's response is expected to be a JSON array of short strings (each a subplot thread description), akin to how subplots per chapter were generated in the old structure service (which returned JSON array of subplot ideas) [14]. We parse the LLM response (using a robust JSON extraction method like `ExtractJsonPayload` as seen in StructureOutliner [15]) and assign it to the act's `"subPlots"` field. If the project's `depthEnabled` flag is false (meaning the user doesn't want deep subplot generation), we can either skip the LLM call or simply leave `subPlots` as an empty array for each act (similar to how StructureOutlinerService inserted empty subPlots when depth was disabled [16]).

- **Update outline progress**: After processing all acts, we set `outline["outlineProgress"] = "SubPlotsDefined"` and write the outline back to disk. We also bump the outline's schemaVersion number, as other steps do, to reflect that the outline content has been advanced [17]. For example:

```
outline["outlineProgress"] = OutlineProgress.SubPlotsDefined.ToString();
outline["header"]["schemaVersion"] = outline["header"]
```

```
["schemaVersion"]!.Value<int>() + 1;
File.WriteAllText(outlinePath, outline.ToString());
```

This marks the completion of the SubPlotsDefined phase.

At the end of this service, the outline now contains a structured breakdown of acts and their subplot threads. All original data (premise, genre, characters, etc.) is still present, and we have added chapters placeholders and an acts/subplots structure, setting the stage for the next phase.

4. Implementing BeatExpanderService (BeatsExpanded Phase)

Next, we implement **BeatExpanderService** to handle the BeatsExpanded phase, which will **assign detailed beats to each chapter (and set titles/summaries)**. This essentially replaces what the old StructureOutliner did for chapters, but we will leverage the act structure and subplot threads for more context.

- **Preconditions**: Ensure the outline is at SubPlotsDefined phase before proceeding (i.e., `outline["outlineProgress"] == "SubPlotsDefined"`). The service will load the outline JSON (and possibly the project JSON again if needed for any remaining settings or audience inclusion).

- **Chapter outline generation**: We need to generate for *each chapter* a title, a summary, a list of beats, and themes. In the old pipeline, StructureOutlinerService did this in one big LLM call for all chapters [18] [19]. For better control and to incorporate act-specific context, our approach can segment this by acts:

- We determine how the total chapters are divided among the acts. For example, if the story has 20 chapters and 5 acts (as in the sample outline), we might allocate roughly 4 chapters per act. We can distribute chapters as evenly as possible across the acts in order. (Prologue and epilogue, if present, are outside the acts and will be handled separately.)

- For each act (1, 2, 3, ...), we gather its summary and subplot threads from the outline (from the data produced in SubPlotDefinerService). Then we prompt the LLM to outline the chapters in that act. For instance:

```
"You are outlining chapters for Act <N> of a novel.\nGenre: <Genre>\nAct
<N> summary: <act summary>...\nSubplots in this act: <list of subplot
ideas>.\nWe need to create <K> chapters for this act. Provide a JSON array
of <K> objects, each with:\n number, title, summary (1-3 sentences), beats
(3-6 one-line beats), and themes (1-3 words).\nEnsure the chapters reflect
the act's events and incorporate the subplot threads where appropriate."
```

We include the genre and audience as needed (to maintain tone consistency), and crucially, we feed in the act's synopsis and subplot ideas to guide the content of chapters. This helps preserve story continuity – for example, chapter outlines will naturally include characters and events mentioned in the act summary or subplots. The model should return a JSON array of chapter outlines for that act. We parse it and then assign the returned titles, summaries, beats, and themes to the actual chapter

objects in the outline (filling in the placeholders created earlier). We also assign correct chapter `"number"`s. If the LLM does not return the numbers or uses a local numbering (e.g., 1, 2, 3 per act), we override or adjust them to the correct global chapter numbers for the outline.

- We repeat the above for each act. This effectively breaks the work into smaller chunks per act, which is easier for the LLM to handle and yields contextually coherent chapters tied to each act's part of the story.

- **Prologue/Epilogue**: If a prologue (chapter 0) is needed, we generate it separately. We can prompt the LLM with the story premise or Act 1 context to create a prologue chapter outline. For example:

```
"You are outlining the Prologue of the novel.\nGenre: <Genre>\nPremise:
<short premise>...\nProvide a JSON object for the prologue with title,
summary, beats (3-6), and themes."
```

  We parse the single-object JSON and fill chapter 0's fields. Similarly, for an epilogue (chapter N+1), we can use the final act's outcome as context (or the entire storyArc) to prompt an epilogue outline that wraps up the story. This ensures those sections are not omitted. In the sample outline, for instance, the epilogue (chapter 21) summary clearly follows from the Act 5 resolution [11] [12] – our prompting will aim for a similar continuity.

- **Populating the outline**: As each chapter's data is produced, we set the chapter object's fields:

- `"title"` – a brief chapter title from the LLM.
- `"summary"` – a few sentences summarizing the chapter.
- `"beats"` – an array of the key beats or events in the chapter (usually 3–6 items).
- `"themes"` – an array of 1–3 themes or motifs for the chapter.

This matches the structure defined in the schema [20] and what the old StructureOutliner produced. For reference, a chapter entry after this phase should look like the sample outline's chapters, e.g. chapter 1 in the sample has a title, summary, 5 beats, and 2 themes [21] [22].

- **Update progress**: Once all chapters have been expanded with beats (and titles, summaries, themes), we update `outline["outlineProgress"] = "BeatsExpanded"` and increment the schemaVersion in the header (just as done in previous steps). We then save the outline to disk.

After BeatExpanderService runs, the outline file contains a fully detailed structure of the novel: every chapter (including prologue/epilogue if any) has its title, summary, beats, themes, and each chapter falls within an act that has defined subplot threads. Essentially, we have achieved what was previously `"StructureOutlined"` (and more), but in a more granular and clear way.

Importantly, **no essential data is lost** in this process – the premise, genre, targetAudience, and characters from earlier phases remain in the outline. In fact, by using the act summaries and subplot threads, we ensure these earlier elements influence the chapter outlines. For example, character names introduced in the CharactersOutlined phase will likely appear in the act summaries (the storyArc) and thus in the chapter beats. The final outline is rich with all details needed for the drafting stage.

5. Updating the CLI Verbs

We add two new CLI verbs to trigger the above services, and remove the old one:

- `novelist outline subplots`: This will correspond to running the SubPlotDefinerService. In *Program.cs*, we add a case in the command dispatch switch:

  ```
  ("outline", "subplots") => RunSubPlots(args[2..], client, includeAud),
  ```

  and implement `RunSubPlots` similarly to the existing outline commands. It should require an `--outline` argument (path to the outline JSON) and optionally accept `--model`. Inside, we instantiate `new SubPlotDefinerService(client, includeAud)` and call an async method like `.DefineSubPlotsAsync(outlinePath, model).Wait()`, analogous to how `RunStruct` was calling `DefineStructureAsync` [23]. After the service completes, we might call `SaveSnapshot(o)` (consistent with other steps) and print a success message ("Subplots defined.").

- `novelist outline beats`: Similarly, add a case:

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

  and implement `RunBeats` to invoke `BeatExpanderService`. This will call something like `.ExpandBeatsAsync(outlinePath, model).Wait()`, then snapshot and print "Beats expanded." (The method naming could be `ExpandBeatsAsync` or `DefineBeatsAsync` – choose a clear name and implement accordingly).

By adding these, we enable users (or scripts) to run the new phases via the CLI. We also remove the `"outline define-structure"` case and its handler, as mentioned. After our changes, the relevant portion of the CLI dispatch might look like:

```
return (args[0].ToLowerInvariant(), args[1].ToLowerInvariant()) switch
{
    ("outline", "create")            => RunCreate(...),
    ("outline", "expand-premise")    => RunExpand(...),
    ("outline", "define-arc")        => RunArc(...),
    ("outline", "define-characters") => RunChars(...),
    ("outline", "subplots")          => RunSubPlots(...),
    ("outline", "beats")             => RunBeats(...),
    ("outline", "mark-premise-expanded") => RunMark(...),
    _ => Help()
};
```

This aligns with the requested verbs. We pass the `includeAud` flag to the new services just as we did for arc and structure (so that if `--no-audience` is specified, the prompts won't include the audience line).

6. Updating the Batch Script

The **run_novelist_live.bat** script needs to call the new phases in sequence and omit the old structure step. We modify it as follows:

- Remove the lines that echoed and ran the structure step [6].
- Insert new steps for subplots and beats after the characters step. For example:

```
REM -------- STEP 5: Define subplots ----------------------------------------
echo ------------------------------------------------------------------------
echo STEP 5: Define subplots
echo ------------------------------------------------------------------------
dotnet run --project "%CLI%" -- outline subplots --outline "%OUTLINE%" --model
%MODEL% --live -s
if errorlevel 1 goto :error

REM -------- STEP 6: Expand beats -------------------------------------------
echo ------------------------------------------------------------------------
echo STEP 6: Expand beats
echo ------------------------------------------------------------------------
dotnet run --project "%CLI%" -- outline beats --outline "%OUTLINE%" --model
%MODEL% --live -s
if errorlevel 1 goto :error
```

Each uses the `--live -s` flags similarly, ensuring we use the live model and save snapshots of each stage (so intermediate outline files like *_SubPlotsDefined.json and* _BeatsExpanded.json will be saved for debugging, following the snapshot naming mechanism in Program.cs [24]).

- Adjust the step numbering comments accordingly (so the script prints the correct step numbers and descriptions).

After this change, running the batch will execute: create -> expand-premise -> define-arc -> define-characters -> define-subplots -> expand-beats. If all steps succeed, it will output a final outline with outlineProgress "BeatsExpanded". The script will no longer attempt the removed structure step.

7. Preserving Data for Downstream Chapter Generation

A critical aspect of these changes is that **no important data is lost** for the subsequent chapter drafting process. We verify the following:

- The **premise**, **story genre**, **target audience**, and **characters** remain in the outline JSON. Our new services do not remove or overwrite these fields. In fact, the OutlineBuilder's initial skeleton includes premise, storyGenre, targetAudience, etc. [25], and we only add new fields (acts, subplots, chapter details) on top of it. The characters generated in CharactersOutlinerService stay under `outline["characters"]` [26].

- We carry forward the **author preset influence**: The CharactersOutlinerService already applied the `famousAuthorPreset` if provided (in character generation prompts [27]), and the preset value is stored in the project. The drafting step (when writing full chapters) uses the project's famousAuthorPreset to emulate style [28]. Our outline pipeline changes do not affect this, since we neither remove the preset from the project nor ignore it in drafting. The new outline phases themselves don't explicitly need the author preset, so this data remains safely in the project file for later use.

- **Prologue/Epilogue**: By reading `includePrologue` and `includeEpilogue` from the project and explicitly creating those chapter entries, we ensure that if the user requested a prologue/epilogue, the outline will contain them. This is important because the drafting stage checks the project flags and expects corresponding chapters in the outline. For example, ChapterDraftService looks for chapter 0 if includePrologue is true [29]. With our changes, if `includePrologue` is true, there will indeed be a chapter 0 in the outline (filled with beats and summary by BeatsExpanded phase), so the drafting process can proceed correctly for it. The same goes for the epilogue chapter.

- **Themes and beats**: We continue to include **themes** for each chapter, just as the old structure step did (1–3 themes per chapter). These are included in the LLM prompt and output for chapter outlining. The final outline will have a `"themes"` array in each chapter object [30] [22]. These themes are passed into the chapter drafting prompt (they appear as a line in the prompt, e.g. "Themes: mystery, fear" [31]), so preserving them is beneficial for guiding tone in generation. Our BeatsExpanderService ensures to gather theme suggestions for each chapter.

- **Premise and Arc continuity**: Although the premise and storyArc are not explicitly included in every prompt, our multi-phase approach inherently uses them. The ArcDefined phase stored the high-level act summaries (storyArc text) in the outline [32], and our SubPlotDefiner uses those summaries. This means the essence of the premise and arc flows into the subplots and then into chapter details. For instance, key plot points from the premise (e.g. the mysterious fog and Old Man Cormac in the sample premise [33]) are present in the act summaries [32] and thus will influence chapter generation. Indeed, in the sample final outline, characters and events from the premise show up in chapter summaries and beats (Eleanor, Cormac, the fog's curse, etc.), indicating the chain of data was preserved.

In summary, the outline produced at the end of the BeatsExpanded phase contains everything needed for downstream chapter drafting: each chapter's summary and beats (required by the draft prompt [34]), themes (for style/flavor), plus the global context (premise, genre, audience, characters) remains either in the outline or accessible via the linked project file. We have added detail (acts and subplot info) without losing anything.

8. Finalizing Changes and Creating a Pull Request

All the above modifications will be implemented in the codebase (across the *Novelist.OutlineBuilder* services, *Novelist.Cli* Program, schema files, and batch script). We will test the new outline pipeline to ensure each phase transitions correctly and the final JSON matches expectations (it should resemble the previous structure's output, but now with an `"acts"` section and possibly per-act subplots, and outlineProgress ending at "BeatsExpanded").

Finally, we commit all changes in a cohesive commit (since these changes are interdependent) with a message such as **"Add SubPlotsDefined and BeatsExpanded phases to outline pipeline; remove StructureOutlinerService"**. Then we open a pull request targeting the `main` branch with a detailed description of these changes and their rationale (reflecting the points above). The PR will link to this issue for context.

Once merged, the outline-building process will proceed through the new Subplots and Beats phases, and the outdated structure phase will be fully retired, aligning the application with the requested new design.

Sources:

- Code references from the `thenovelist` repository illustrating current implementation and schema:
- OutlineProgress enum definition [1] and usage in helpers [2].
- Outline schema v1 (excerpt of outlineProgress enum) [3].
- Program.cs CLI command handling [4] [5].
- Batch script steps [6].
- StructureOutlinerService logic for structure and subplots [35] [14].
- Sample outline JSON showing final structure with chapters, beats, themes, etc. [36] [21].
- Project schema (prologue/epilogue flags) [37] and draft building usage of outline data [38].

---

[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] [20] outline.schema.v1.json
https://github.com/bilols/thenovelist/blob/3e7a893f93e2cc7ee7cfa1958fe045abdc55414b/tools/outline.schema.v1.json

[4] [5] [23] [24] Program.cs
https://github.com/bilols/thenovelist/blob/3e7a893f93e2cc7ee7cfa1958fe045abdc55414b/src/Novelist.Cli/Program.cs

[6] run_novelist_live.bat
https://github.com/bilols/thenovelist/blob/3e7a893f93e2cc7ee7cfa1958fe045abdc55414b/run_novelist_live.bat

[7] [13] [14] [15] [16] [18] [19] [35] StructureOutlinerService.cs
https://github.com/bilols/thenovelist/blob/3e7a893f93e2cc7ee7cfa1958fe045abdc55414b/src/Novelist.OutlineBuilder/StructureOutlinerService.cs

[8] [25] OutlineBuilderService.cs
https://github.com/bilols/thenovelist/blob/3e7a893f93e2cc7ee7cfa1958fe045abdc55414b/src/Novelist.OutlineBuilder/OutlineBuilderService.cs

[9] [27] CharactersOutlinerService.cs
https://github.com/bilols/thenovelist/blob/3e7a893f93e2cc7ee7cfa1958fe045abdc55414b/src/Novelist.OutlineBuilder/CharactersOutlinerService.cs

10 11 12 21 22 26 30 32 33 36 outline_20250703212612.json

https://github.com/bilols/thenovelist/blob/3e7a893f93e2cc7ee7cfa1958fe045abdc55414b/tests/samples/outline_20250703212612.json

17 ArcDefinerService.cs

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

28 31 34 38 PromptBuilder.cs

https://github.com/bilols/thenovelist/blob/3e7a893f93e2cc7ee7cfa1958fe045abdc55414b/src/Novelist.DraftBuilder/PromptBuilder.cs

29 ChapterDraftService.cs

https://github.com/bilols/thenovelist/blob/3e7a893f93e2cc7ee7cfa1958fe045abdc55414b/src/Novelist.DraftBuilder/ChapterDraftService.cs

37 project.schema.v1.json

https://github.com/bilols/thenovelist/blob/3e7a893f93e2cc7ee7cfa1958fe045abdc55414b/tools/project.schema.v1.json