C03 XML Prompting & Semantic Anchors

Conceptual Advanced

Structuring complex prompts for reproducible results

Why XML works

Claude was trained on massive corpora that include XML. When you structure a prompt with tags, you create unambiguous delimiters that separate the instruction, context, reference code, and constraints. The result: Claude interprets each section exactly in the role you assign it, without having to infer what is instruction and what is context.

XML structure is relevant for complex or reusable tasks. For a one-line fix, it is unnecessary noise.

Basic structure

```
<instruction>
Extract the validation logic into a separate
module
</instruction>

<context>
Express service, Node.js 20, no external
dependencies
</context>

<code_example>
// Existing pattern to preserve
function validateEmail(email: string): boolean {
  return /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email)
}
</code_example>

<constraints>
- Do not modify public function signatures
- Maintain compatibility with existing tests
</constraints>

<output>
New file src/validation/index.ts with exports
</output>
```

Reusable tag catalog

Tag	Usage
`<instruction>`	The main task
`<context>`	Stack, system state, history
`<code_example>`	Reference pattern to follow
`<constraints>`	What must not be modified
`<output>`	Expected format and content
`<state>`	Current state (bug, diff, logs)
`<task>` + `<subtask>`	Hierarchical decomposition

Semantic Anchors in code

Anchors are comment markers that allow Claude to reference specific areas of code without reading entire files.

```
// ANCHOR: auth-flow
export async function authenticate(token: string)
{
  // ...
}
// END-ANCHOR: auth-flow

// ANCHOR: token-validation
function validateJWT(token: string): Claims {
  // ...
}
// END-ANCHOR: token-validation
```

In your prompt: “Modify only the ANCHOR: auth-flow section to add expired token handling.” Claude knows exactly where to intervene without reading the entire file.

When to use XML

Case	Use XML?
Simple single-line request	No
Bug to fix in one function	No
Multi-file implementation	Yes
Reusable prompt in a command	Yes
Review with precise criteria	Yes

Integration with CLAUDE.md

You can define the project XML conventions in `CLAUDE.md` to standardize team prompts:

```
## XML Prompt Conventions
Project tags: <api_design>, <accessibility>,
<perf_budget>
For features: always include <context> and
<constraints>
```

All team members then use the same tag vocabulary, making shared prompts (in `.claude/commands/`) consistent and understandable.