🧩 MACROS.md – Swift Macros in SpecificationKit

This document describes the role, benefits, and design plans for using **Swift macros** in the SpecificationKit library to enhance developer experience and enable declarative specification logic.

🎯 Purpose of Macros

Goal	Benefit
🔽 Reduce boilerplate	Auto-generates composite specifications
📜 Declarative syntax	Enables `@specs(...)` over manual `.and().or()` chaining
🔐 Compile-time safety	Ensures only valid specifications are composed
🧩 Extendability	Allows high-level, modular rule declarations

🔧 Macro Type: `@specs` (Attached Macro)

Example Usage

```swift
@specs(
  TimeSinceEventSpec(minimum: 10),
  MaxCountSpec(limit: 3),
  CooldownIntervalSpec(interval: .days(7))
)
struct CompositeSpec: Specification { }
```

What It Generates

```swift
struct CompositeSpec: Specification {
  private let composite: AnySpecification<EvaluationContext>

  init() {
    composite = AnySpecification(
      TimeSinceEventSpec(minimum: 10)
        .and(MaxCountSpec(limit: 3))
        .and(CooldownIntervalSpec(interval: .days(7)))
    )
  }

  func isSatisfiedBy(_ context: EvaluationContext) -> Bool {
    composite.isSatisfiedBy(context)
  }
}
```

🧪 Compile-time Validation

The macro can:

- Verify all elements conform to `Specification`
- Ensure consistent `Context` types
- Fail gracefully at compile time with clear diagnostics

🔘 Optional: Expression Macros

Future possibility for inline usage:

```
@Satisfies(using: #specs(
  TimeSinceEventSpec(minimum: 10),
  MaxCountSpec(limit: 3)
))
var shouldProceed: Bool
```

🛠️ Implementation Plan

- [ ] Create new macro target: `SpecificationKitMacros`
- [ ] Implement attached macro: `@specs`
- [ ] Use `swift-syntax` to parse initializer arguments
- [ ] Generate `init()` + `isSatisfiedBy(_:)`
- [ ] Write tests using `MacroTesting`

💡 Ideas for Future Macros

Macro	Purpose
`@AutoContext`	Auto-generates contextProvider from known sources
`@specsIf(condition:)`	Conditional spec generation
`#composed(...)`	Inline expression-based composition macro
`@deriveSpec(from:)`	Generates specs from model annotations

✅ Summary

Swift Macros in SpecificationKit will:

- Make specifications **easier to write**
- Reduce **boilerplate**
- Improve **safety and clarity**
- Enable **new forms of composition** and configuration

The `@specs(...)` macro is the first step toward a powerful declarative spec DSL.