Configuring Rules in Windsurf IDE: A Comprehensive Guide

Understanding Windsurf Rules Configuration Structure

Windsurf's powerful configuration system allows you to define rules that guide how the AI assistant (Cascade) behaves when generating or modifying code [1] [2]. These rules create a personalized development environment that understands your coding standards, architectural preferences, and project requirements [3] [4].

File Locations and Access Methods

Accessing the Rules Configuration Interface

The primary way to configure rules in Windsurf is through the Cascade interface's customization panel [1] [2] [5]:

1. Open the Windsurf IDE
2. Click on the "Manage" button in the Cascade panel
3. Select "Customizations" to access the rules configuration interface [5] [2]

This interface provides access to both global and workspace-specific rules [1] [5].

File Structure and Locations

Rules in Windsurf are stored as markdown files in specific locations [1] [2] [6]:

Global Rules:

- Location: `~/.codeium/windsurf/global_rules.md` (Linux/macOS) or `%USERPROFILE%\.codeium\windsurf\global_rules.md` (Windows) [6] [7]
- Purpose: Applied across all Windsurf IDE instances [1] [5]

Workspace/Project Rules:

- Location: `.windsurf/rules/` directory in your project folder [6] [8]
- Purpose: Applied only to the specific project/workspace [1] [2] [5]

When you create rules through the Customizations interface, Windsurf automatically creates these directories and files as needed [2] [6].

Rule File Format and Structure

Windsurf rules are stored in markdown files with a straightforward structure [1] [8] [6]:

```
# Rule Title

This is the content of your rule that guides Cascade's behavior.
You can include multiple paragraphs, code examples, and formatting.

- Use bullet points for lists of requirements
- Include specific examples when possible
```

Rules should be concise and specific, as there are character limits [1] [9]:

- Individual rules must stay under 6,000 characters [1] [8]
- Combined rules (global + workspace) have a limit of 12,000 characters [1] [8]

Rule Activation Modes

Windsurf supports four activation modes that determine when rules are applied [1] [2] [6]:

1. **Always On**: Rules are automatically applied to every prompt to Cascade [1] [6]
2. **Manual**: Rules are activated only when explicitly mentioned using @ruleName in the prompt [1] [6] [8]
3. **Model Decision**: Cascade decides whether to apply the rule based on context and a provided description [1] [6] [8]
4. **Glob/Pattern-Based**: Rules are applied only to files that match specific patterns (e.g., `*.js`, `src/**/*.ts`) [1] [8] [6]

To set the activation mode, use the dropdown menu in the Customizations interface when creating or editing rules [1] [2] [8].

Examples of Global Rules

Global rules typically focus on general coding principles, security guidelines, and team-wide standards [3] [4]. Here are examples:

Coding Style Example

```
# Coding Style Standards

Always follow these principles when generating or modifying code:

1. Use consistent indentation (2 spaces for JavaScript, 4 spaces for Python)
2. Follow camelCase naming for variables and functions in JavaScript
3. Use PEP 8 style guidelines for Python code
4. Include appropriate comments for complex logic
5. Keep functions small and focused on a single responsibility
6. Use early returns to reduce nesting depth
```

```
For security:
- Sanitize all user inputs
- Use parameterized queries for database operations
- Never include hardcoded credentials
```

Library Preferences Example

```
# Library Preferences

Use Tailwind CSS for styling instead of raw CSS or other CSS frameworks.
For state management in React applications, prefer React Context API for simpler applicat
Use Jest for testing JavaScript and TypeScript code.
```

Examples of Project-Specific Rules

Workspace rules focus on framework conventions, styling preferences, and project-specific patterns [3] [4] [8]. Here are examples:

Framework-Specific Rules

```
# Next.js Best Practices

- Use the App Router for all new components and pages
- Place reusable components in the components directory
- Place page-specific components in the app directory
- Use server components by default, only use client components when necessary
- Follow the naming convention of page.tsx for route pages
- Use layout.tsx for shared layouts
- Place API routes in the app/api directory
- Use SWR for data fetching on the client side
```

Tech Stack Specification

```
# Project Tech Stack

This project uses the following technologies:

Frontend:
- React 18
- Next.js 14
- TypeScript 5
- Tailwind CSS 3
- SWR for data fetching

Backend:
- Node.js 20
- Express 4
- PostgreSQL 15
- Prisma ORM
- Jest for testing

Infrastructure:
- Docker for containerization
- AWS for hosting
- GitHub Actions for CI/CD
```

Context Priming and First Prompts

When starting a new project, it's helpful to provide Cascade with context about your project structure and goals [4] [10]. Here's an example of a rule for context priming:

```
# Project Context

This project is an e-commerce platform with the following features:
- User authentication and profile management
- Product catalog with categories and search functionality
- Shopping cart and checkout process
- Order management
- Admin dashboard for inventory management

The application follows a monorepo structure with:
- `/apps/web` - Customer-facing Next.js application
- `/apps/admin` - Admin dashboard (React)
- `/apps/api` - Backend API (Express + TypeScript)
- `/packages/ui` - Shared UI components
- `/packages/core` - Shared business logic

Follow these architectural patterns:
- Use React Query for data fetching in frontend applications
- Implement form handling with React Hook Form
- Use zod for schema validation
- Follow repository pattern for database access
- Implement command/query separation for business logic
```

Pattern-Based Rules Example

For configuring glob-pattern rules that apply to specific file types [1] [8] [6]:

```
# TypeScript Standards

- Use explicit type annotations for function parameters and return types
- Prefer interfaces over types for object definitions
- Use enums for fixed sets of values
- Avoid using 'any' type
- Use optional chaining and nullish coalescing operators
- Follow ESLint configuration
```

Set the activation mode to "Glob" and specify the pattern as `**/*.ts,**/*.tsx` to apply this rule to all TypeScript files [1] [8].

MCP Server Configuration

Model Context Protocol (MCP) servers extend Windsurf's capabilities by connecting external tools and services [11] [12] [7]. Configure MCP servers in:

- Location: `~/.codeium/windsurf/mcp_config.json` (Linux/macOS) or `%USERPROFILE%\.codeium\windsurf\mcp_config.json` (Windows) [7] [12]

Example configuration:

```
{
  "mcpServers": {
    "github": {
      "command": "npx",
      "args": ["-y", "github-mcp-server"],
      "env": {
        "GITHUB_TOKEN": "your_github_token_here"
      }
    },
    "postgresql": {
      "command": "npx",
      "args": ["-y", "postgresql-mcp-server"],
      "env": {
        "PG_CONNECTION_STRING": "postgresql://user:password@localhost:5432/database"
      }
    }
  }
}
```

Best Practices for Rules Configuration

To get the most out of Windsurf's rules system [1] [4] [10]:

1. **Start with community examples**: Visit windsurf.run to discover rules other developers are using [5] [13]
2. **Be specific and concise**: Keep rules focused on specific patterns or requirements [1] [8]
3. **Layer your rules appropriately**: Use global rules for team-wide standards and workspace rules for project-specific patterns [1] [2] [5]
4. **Use XML-style tags for organization**: Group related guidelines in simple tags like `<coding_guidelines>...</coding_guidelines>` [6] [4]
5. **Test and refine**: Continuously improve your rules based on Cascade's outputs [5] [10]
6. **Use context handovers**: For complex projects, create comprehensive context in your rules file [10] [4]

Rules for Different Development Workflows

Different types of projects benefit from specialized rules [13] [10]:

For Frontend Development

```
# Frontend Development Rules

- Use functional components with hooks instead of class components
- Implement proper keyboard navigation and ARIA attributes for accessibility
- Follow atomic design principles for component organization
- Use CSS modules or styled-components for styling
- Optimize images and use proper lazy loading techniques
- Implement responsive designs using mobile-first approach
```

For Backend Development

```
# Backend Development Rules

- Follow RESTful API design principles
- Use proper error handling and status codes
- Implement input validation for all API endpoints
- Use environment variables for configuration
- Implement proper logging
- Use dependency injection for better testability
- Follow the principle of least privilege for security
```

By configuring these rules effectively, you can transform Windsurf from a capable IDE into a highly personalized development environment that understands your team's standards and project requirements [3] [1] [4].

⁂

1. https://www.youtube.com/watch?v=D3uuCeOLfJA
2. https://www.youtube.com/watch?v=PCyw5nRVzYw
3. https://www.codecademy.com/article/how-to-build-an-app-with-windsurf-ai
4. http://blog.nilenso.com/blog/2025/05/29/ai-assisted-coding/
5. https://www.youtube.com/watch?v=xc9kktNRfaY
6. https://www.reddit.com/r/ArtificialInteligence/comments/1kw16yi/a_comprehensive_list_of_agentrule_files_do_we/
7. https://github.com/eyaltoledano/claude-task-master
8. https://github.com/topics/cursor-rules
9. https://github.com/johnlindquist/dotagent
10. https://news.ycombinator.com/item?id=44163063
11. https://deeplearning.fr/maximizing-your-claude-max-subscription-complete-guide-to-automated-workflows-with-claude-code-and-windsurf/

12. https://docs.snyk.io/snyk-cli/developer-guardrails-for-agentic-workflows/quickstart-guides-for-mcp/windsurf-guide
13. https://clickup.com/blog/windsurf-alternatives/