RuleKeeper

Detailed Policy Guide

Parallex Bank IT Coding Standards

This document provides comprehensive documentation for all RuleKeeper coding standard policies. Each policy includes detailed descriptions, regex patterns, good and bad examples, and fix hints.

Version 1.0.0

Table of Contents

Introduction

RuleKeeper is a Policy-as-Code tool designed to scan source code and validate compliance with organizational coding standards defined in YAML configuration files.

YAML Configuration Options

Each rule in the YAML configuration supports the following options:

- enabled: true/false - Activate or deactivate the rule
- skip: true/false - Skip this rule during scanning
- pattern: Regex pattern that valid code should match
- anti_pattern: Regex pattern that indicates a violation
- custom_validator: Reference to a custom validation function
- prebuilt: Reference to a prebuilt policy template
- parameters: Additional configuration parameters

Severity Levels

CRITICAL	Must be fixed immediately - security or compliance risk. Blocks deployment.
HIGH	Should be fixed before deployment. Requires justification to override.
MEDIUM	Should be addressed in the current sprint. Warning only.
LOW	Best practice recommendation. Informational.

Naming Conventions

CS-NAME-001 Class/Interface Naming

HIGH

Classes and Interfaces must use PascalCase

Patterr

Good Example:

```
AccountService, IAccountRepository, TransactionHandler
```

Bad Example:

```
accountservice, account_service, iAccountRepo
```

Fix: Rename to start with uppercase letter, e.g., 'AccountService'

CS-NAME-002 Method Naming

HIGH

Methods must use PascalCase

Patterr

Good Example:

```
GetAccountBalance, ProcessPayment, ValidateInput
```

Bad Example:

```
getbalance, process_payment, validateInput
```

Fix: Rename to start with uppercase letter

CS-NAME-003 Variable/Field/Parameter Naming

HIGH

Variables, fields, and parameters must use camelCase

Patterr

Good Example:

```
accountId, transactionAmount, userName
```

Bad Example:

```
AccountId, transaction_amount, UserName
```

Fix: Rename to start with lowercase letter

CS-NAME-004 Constant Naming

HIGH

Constants must use UPPER_SNAKE_CASE

Patterr

Good Example:

```
MAX_RETRY_COUNT, DEFAULT_TIMEOUT, API_VERSION
```

Bad Example:

```
maxRetryCount, defaultTimeout, ApiVersion
```

Fix: Rename using uppercase with underscores

CS-NAME-005 Private Field Naming HIGH

Private fields must use _camelCase (underscore prefix)

Pattern

Good Example:
```
_accountRepository, _logger, _transactionService
```
Bad Example:
```
AccountRepo, accountRepository, _AccountRepo
```
Fix: Rename with underscore prefix and lowercase

CS-NAME-006 Async Method Naming HIGH

Async methods must end with 'Async' suffix

Pattern

Good Example:
```
GetAccountBalanceAsync, ProcessPaymentAsync
```
Bad Example:
```
GetAccountBalance (for async methods)
```
Fix: Add 'Async' suffix to async method names

CS-NAME-007 Interface Naming HIGH

Interfaces must be prefixed with 'I'

Pattern

Good Example:
```
IAccountService, ITransactionRepository, ILogger
```
Bad Example:
```
AccountServiceInterface, AccountService (for interfaces)
```
Fix: Add 'I' prefix to interface names

CS-NAME-008 Request/Response DTO Naming HIGH

Request and Response DTOs must end with Request or Response suffix

Pattern

Good Example:
```
TransferRequest, AccountResponse, LoginRequest
```
Bad Example:
```
TransferDTO, AccountData, LoginPayload
```
Fix: Add 'Request' or 'Response' suffix

CS-NAME-009 Boolean Variable Naming MEDIUM

Boolean variables should use is/has/can/should prefixes

CS-NAME-010 Event Handler Naming MEDIUM

Event handlers should follow 'On' + EventName pattern

File & Project Organization

CS-FILE-001 **One Class Per File** MEDIUM

Each file should contain only one class

CS-FILE-002 **File Name Matches Class** HIGH

File name must match the class name it contains

CS-FILE-003 **Feature-Based Organization** MEDIUM

Group files logically by feature, not layer (vertical slicing)

CS-FILE-004 **Namespace Matches Folder Structure** MEDIUM

Namespace should reflect the folder structure

Method Design & Readability

CS-METHOD-001 **Single Responsibility** HIGH

Methods should be small and do one thing

CS-METHOD-002 **Method Length** MEDIUM

Keep method length at or below 30 lines

CS-METHOD-003 **Parameter Count** MEDIUM

Avoid long parameter lists - use DTOs instead

CS-METHOD-004 **Cyclomatic Complexity** MEDIUM

Methods should have low cyclomatic complexity

CS-METHOD-005 **No Nested Ternary** MEDIUM

Avoid nested ternary operators

Secure Coding Practices

CS-SEC-001 **Parameterized Queries** CRITICAL

Never concatenate SQL or user inputs - use parameterized queries

Good E

```
cmd.Parameters.AddWithValue("@id", accountId);
```

Bad Example:

```
"SELECT * FROM Accounts WHERE Id = '" + accountId + "'"
```

Fix: Use parameterized queries with @parameters

CS-SEC-002 **Input Validation** CRITICAL

Always validate user input

CS-SEC-003 **Log Sanitization** CRITICAL

Sanitize logs - no sensitive data (PIN, password, token)

Good E

```
_logger.LogInformation("Transfer for {AccountId}", accountId);
```

Bad Example:

```
_logger.LogInformation($"Login with PIN {pin}");
```

Fix: Remove sensitive data from log statements

CS-SEC-004 **Secret Protection** CRITICAL

Use SecureString or data masking for secrets

CS-SEC-005 **Configuration Security** CRITICAL

Protect configuration via Azure Key Vault or AWS Secrets Manager

CS-SEC-006 **No Hardcoded Credentials** CRITICAL

Never hardcode credentials in source code

CS-SEC-007 **XSS Prevention** CRITICAL

Sanitize output to prevent Cross-Site Scripting

CS-SEC-008 **Path Traversal Prevention** CRITICAL

Validate file paths to prevent directory traversal

Exception Handling & Logging

CS-EXC-001 **Meaningful Exception Handling** HIGH

Use try-catch only where you can handle errors meaningfully

CS-EXC-002 **Contextual Logging** HIGH

Log exceptions with context, but not sensitive data

CS-EXC-003 **No Empty Catch Blocks** CRITICAL

Avoid empty catch blocks

Good E

```
catch (Exception ex) { _logger.LogError(ex, "Error"); throw; }
```

Bad Example:

```
catch (Exception) { /* ignore */ }
```

Fix: Log the exception or handle it meaningfully

CS-EXC-004 **Domain Exceptions** MEDIUM

Throw domain-specific exceptions when needed

CS-EXC-005 **No Catch-All Without Rethrow** HIGH

Catching all exceptions should rethrow or terminate

Asynchronous Programming

CS-ASYNC-001 Always Await

HIGH

Always await async calls

CS-ASYNC-002 No Blocking Async

CRITICAL

Don't block async with .Result or .Wait()

Good E

```
var result = await _service.GetAsync();
```

Bad Example:

```
var result = _service.GetAsync().Result;
```

Fix: Use 'await' instead of .Result or .Wait()

CS-ASYNC-003 ConfigureAwait in Libraries

MEDIUM

Use ConfigureAwait(false) in library code

CS-ASYNC-004 Async Void Avoidance

HIGH

Avoid async void except for event handlers

CS-ASYNC-005 Proper Cancellation Token Usage

MEDIUM

Async methods should accept and use CancellationToken

Dependency Injection & SOLID

CS-DI-001 **Depend on Abstractions** HIGH

Depend on interfaces, not concrete types

Good E

```
private readonly ITransactionService _transactionService;
```

Bad Example:

```
private readonly TransactionService _transactionService;
```

Fix: Change type to interface

CS-DI-002 **Use IoC Container** HIGH

Use built-in IServiceCollection or IoC containers

CS-DI-003 **Avoid New in Business Logic** HIGH

Avoid 'new' keyword for dependencies inside business logic

CS-DI-004 **Constructor Injection Only** MEDIUM

Use constructor injection, not property or method injection

CS-DI-005 **Service Lifetime Consistency** HIGH

Ensure consistent service lifetimes in DI registration

Constants & Magic Numbers

`CS-CONST-001` **No Magic Numbers** MEDIUM

Avoid magic numbers or strings in code

`CS-CONST-002` **Use Named Constants** MEDIUM

Use named constants or enums instead of literals

`CS-CONST-003` **No Magic Strings** MEDIUM

Avoid magic strings in code

Data Validation

CS-VAL-001 **DTO Validation** HIGH

Always validate input DTOs using attributes or FluentValidation

CS-VAL-002 **Client and Server Validation** HIGH

Validate both client and server side

CS-VAL-003 **Null Checks** HIGH

Check for null before using objects

CS-VAL-004 **Guard Clauses** MEDIUM

Use guard clauses for parameter validation

Logging Standards

CS-LOG-001	**Structured Logging**	HIGH

Use structured logging

CS-LOG-002	**No Sensitive Data in Logs**	CRITICAL

Never log sensitive data (PIN, password, token)

CS-LOG-003	**Appropriate Log Levels**	MEDIUM

Log at appropriate levels (Info, Warning, Error, Critical)

CS-LOG-004	**Include Correlation ID**	MEDIUM

Include correlation/trace ID in logs for distributed tracing

Code Comments & Documentation

CS-DOC-001	**XML Comments for Public APIs**	MEDIUM

Use XML comments for public APIs

CS-DOC-002	**Comment Why Not What**	LOW

Comment why, not what - avoid redundant comments

CS-DOC-003	**TODO Comments**	LOW

TODO comments should include ticket/issue reference

Immutability & Defensive Coding

CS-IMM-001 Use Readonly MEDIUM

Use readonly for fields that don't change after construction

CS-IMM-002 No Mutable Collections MEDIUM

Avoid exposing mutable collections

CS-IMM-003 Clone External Data MEDIUM

Clone or copy external data inputs

CS-IMM-004 Use Records for DTOs LOW

Consider using records for immutable DTOs

Secure Configuration

CS-CFG-001	**No Secrets in Source**	CRITICAL

No secrets in source code or appsettings.json

CS-CFG-002	**Use Secret Managers**	CRITICAL

Use environment variables or secret managers

CS-CFG-003	**Secure Connection Strings**	HIGH

Connection strings should use integrated security or managed identity

Secure String Handling

CS-STR-001	**No Plain Text Secrets**	HIGH

Avoid keeping secrets as plain strings in memory

CS-STR-002	**Use SecureString**	HIGH

Use SecureString or encrypt secrets in memory

Unit Testing Standards

CS-TEST-001	**Test Naming Convention**	MEDIUM

Use clear test names: MethodName_StateUnderTest_ExpectedBehavior

CS-TEST-002	**Single Assertion**	LOW

Prefer one logical assertion per test

CS-TEST-003	**No External Dependencies**	HIGH

No dependency on external systems in unit tests

CS-TEST-004	**Arrange-Act-Assert Pattern**	LOW

Tests should follow Arrange-Act-Assert pattern

CS-TEST-005	**Test Class Naming**	LOW

Test classes should be named {ClassName}Tests

CORS Configuration

API-CORS-001 Specific CORS Origins CRITICAL

Configure CORS with specific allowed origins, not AllowAnyOrigin

Good E

```
policy.WithOrigins("https://example.com").AllowCredentials();
```

Bad Example:

```
policy.AllowAnyOrigin().AllowAnyMethod();
```

Fix: Specify allowed origins with WithOrigins()

API-CORS-002 No Credentials with Any Origin CRITICAL

AllowCredentials cannot be used with AllowAnyOrigin

API Design

API-REST-001 **RESTful Endpoints** HIGH

Use proper HTTP methods for CRUD operations

API-HTTP-001 **Appropriate Status Codes** HIGH

Return appropriate HTTP status codes

API-VER-001 **API Versioning** HIGH

Implement API versioning in routes

API-RESP-001 **Consistent Response Format** MEDIUM

Use a consistent API response wrapper

API-DOC-001 **Endpoint Documentation** MEDIUM

Document API endpoints with XML comments and response types

Encryption

`API-ENC-001`	**Proper RSA Encryption**	CRITICAL

Use proper RSA encryption with OAEP padding

`API-ENC-002`	**Strong Hashing Algorithms**	CRITICAL

Use SHA-256 or stronger for hashing

Idempotency

API-IDEMP-001 **Idempotency Keys** CRITICAL

Use idempotency keys for financial operations

Authentication & Authorization

API-AUTH-001 Endpoint Authorization

CRITICAL

Protect endpoints with proper authorization

Good E

```
[Authorize(Policy = "BankingCustomer")]
```

Bad Example:

```
[HttpGet("accounts")] // no authorization
```

Fix: Add [Authorize] attribute to secure endpoints

API-AUTH-002 Resource-Level Authorization

CRITICAL

Verify user has access to specific resources

Error Handling

API-ERR-001 **Domain Exception Handling** HIGH

Handle domain-specific exceptions with appropriate responses

API-SAN-001 **Input Sanitization** CRITICAL

Sanitize all user inputs before processing

Rate Limiting

API-RATE-001 **Rate Limiting** **HIGH**

Implement rate limiting on API endpoints

Appendix: YAML Configuration Example

```
# Example rule configuration
coding_standards:
  naming_conventions:
    - id: CS-NAME-001
      name: "Class/Interface Naming"
      description: "Classes must use PascalCase"
      severity: high
      enabled: true
      skip: false
      pattern: "^[A-Z][a-zA-Z0-9]*$"
      anti_pattern: "^[a-z]|_"
      applies_to:
        - classes
        - interfaces
      file_pattern: "**/*.cs"
      custom_validator: null
      message: "Class name must use PascalCase"
      fix_hint: "Rename with uppercase first letter"
      examples:
        good: "AccountService"
        bad: "accountservice"
      tags:
        - naming
        - convention
```

Disabling a Rule

```
    - id: CS-NAME-001
      name: "Class/Interface Naming"
      enabled: false  # Disable this rule
      skip: true      # Also skip during scanning
```

Using Custom Validator

```
    - id: CS-METHOD-002
      name: "Method Length"
      custom_validator: "Validators.ValidateMethodLength"
      parameters:
        max_lines: 50  # Custom parameter
```