Understand Reports

Table of Contents

1. Accepted Risk Register
2. Annotations by Template
3. API Info
4. API Info Table
5. Compliance
6. Control Coupling
7. Data Coupling
8. Data Dictionary
9. Deviation Approval Records
10. Deviation Register
11. Deviation Risk Assessment
12. Guideline Compliance Summary
13. Low Maintainability Classes Table
14. Low Maintainability Functions

Accepted Risk Register

Languages: Any
Targets: CodeCheck
Every suppressed finding that carries a security disposition, worst first.

Security audits ask for a risk-acceptance log: which findings were silenced on purpose, how severe each one is, what weakness class it falls under, and whether the disposition was *Accepted*, a *False Positive*, or something still open. This report lists every suppression whose record carries a **Severity**, **Status** or **CWE** field, ordered by severity, worst first. Suppressions with no security disposition at all are counted so the log cannot silently understate what was silenced.

The records are the ones suppressions already carry (see the Deviation Register).

Annotations by Template

Languages: Any
Targets: Project
Structured-annotation coverage grouped by annotation template.

One page per template shows the value distribution of each closed-set field and a filterable table of every annotation with its field values. Required fields without a value are flagged "(missing)" so classification gaps are easy to find. Freeform annotations get their own page.

Templates are managed in Annotations > Annotation Templates, and values can be assigned in bulk with the und annotate command.

API Info

Languages: Any
Targets: Architectures, Classes, Files, Functions, Objects
Displays Understand API information for the selected entity, architecture, or metric.

This plugin is primarily intended to help developers learn how to use the Understand API. It displays the same information available to Understand plugins when using the API. The output of the plugin can be helpful when developing or debugging a plugin. For more detailed API documentation, please refer to the Understand API documentation, available in the Understand Help menu or online ↗.

See also the API Info Table report which displays the same information in a table format instead of a tree.

_check_dir_contents
API Info

Entity Information as reported by the API

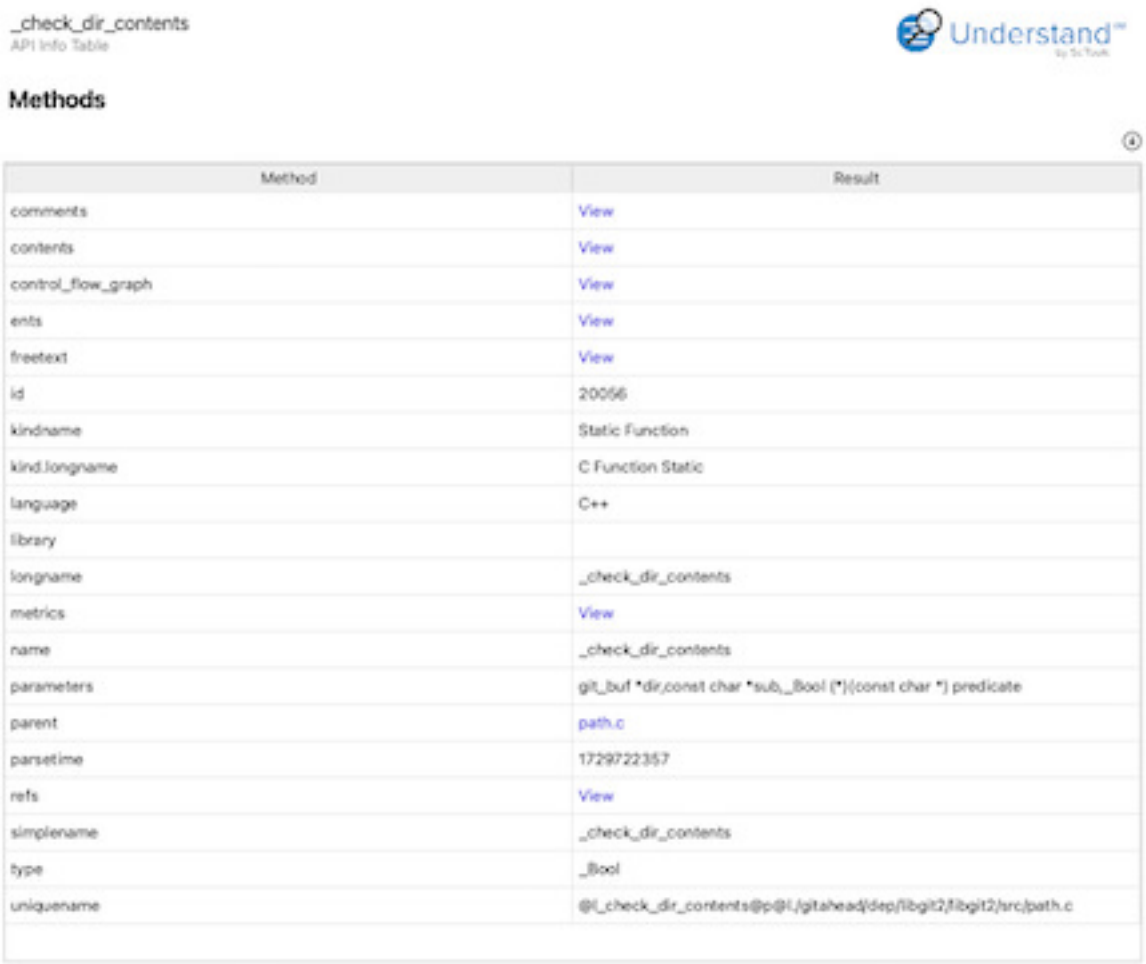


API Info Table

Languages: Any
Targets: Architectures, Classes, Files, Functions, Objects
API Info Table

This plugin is primarily intended to help developers learn how to use the Understand API. It displays the same information available to Understand plugins when using the API. The output of the plugin can be helpful when developing or debugging a plugin. For more detailed API documentation, please refer to the Understand API documentation, available in the Understand Help menu or online ↗.

See also the API Info report which displays the same information in a tree format instead of a table.



_check_dir_contents
API Info Table

Understand™ by SciTools

Methods

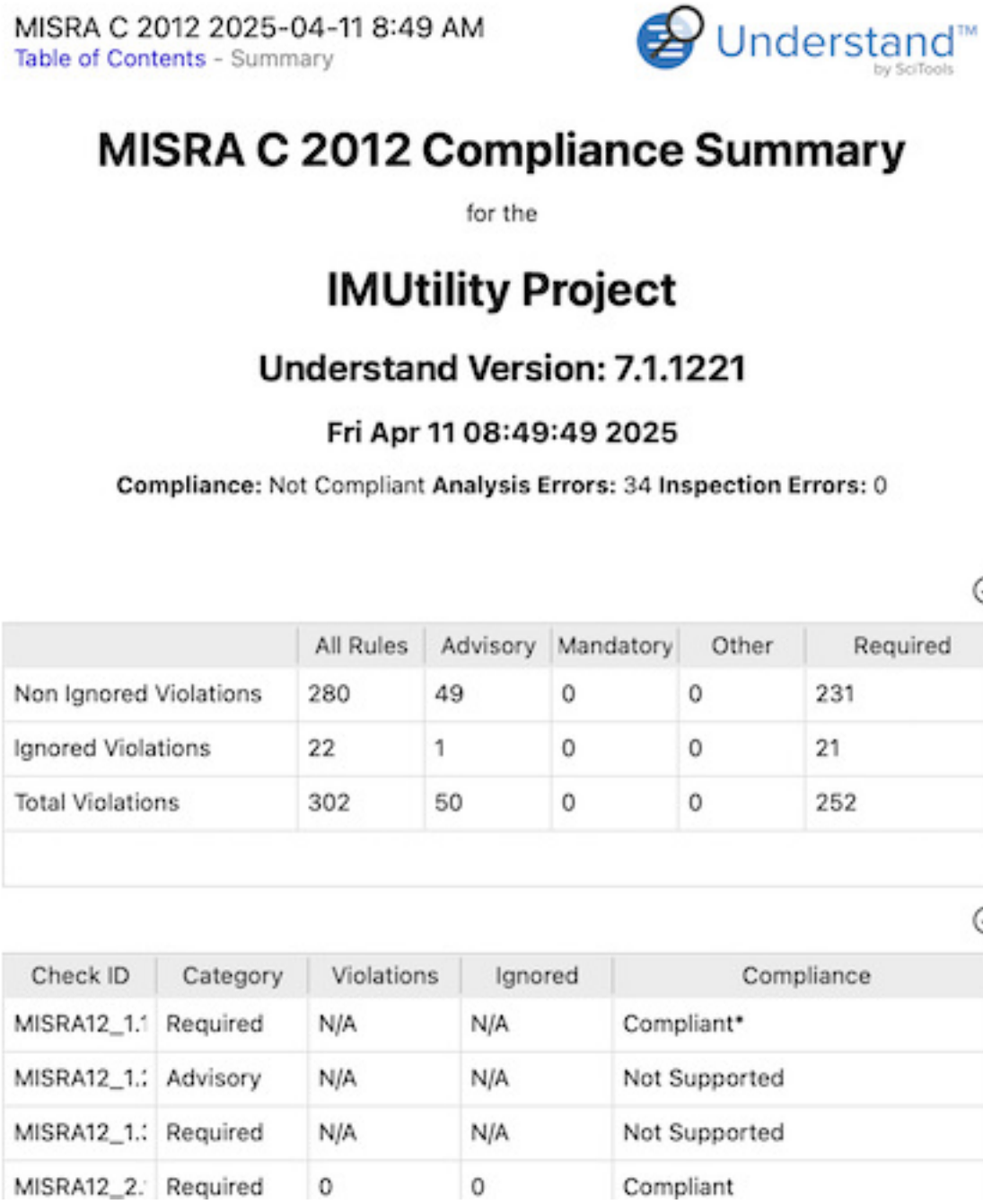
Method	Result
comments	View
contents	View
control_flow_graph	View
ents	View
freetext	View
id	20056
kindname	Static Function
kind.longname	C Function Static
language	C++
library	
longname	_check_dir_contents
metrics	View
name	_check_dir_contents
parameters	git_buf *dir,const char *sub,_Bool (*)(const char *) predicate
parent	path.c
parsetime	1729722357
refs	View
simplename	_check_dir_contents
type	_Bool
uniquename	@l_check_dir_contents@p@l./gitahead/dep/libgit2/libgit2/src/path.c

Compliance

Languages: Any
Targets: CodeCheck
Generates a report of code compliance to a given coding standard.

This plugin generates a report showing the compliance of the code to a chosen coding standard. Users can select report sections:

- **Summary** A table showing the checks in the configuration, the total violations found for each check, the number of those violations that were ignored, and the final "Compliant" or "Not Compliant" status of that check and the overall project.
- **Coverage** Show which of the checks in the configuration are supported by Understand. Also includes a summary table of the number of checks by Category that are supported.
- **Violations** A table listing all the non-ignored violations found
- **Ignores** A table listing all the ignored violations and the justification for ignoring the violation



MISRA C 2012 2025-04-11 8:49 AM
Table of Contents - Summary

MISRA C 2012 Compliance Summary

for the

IMUtility Project

Understand Version: 7.1.1221

Fri Apr 11 08:49:49 2025

Compliance: Not Compliant **Analysis Errors:** 34 **Inspection Errors:** 0

	All Rules	Advisory	Mandatory	Other	Required
Non Ignored Violations	280	49	0	0	231
Ignored Violations	22	1	0	0	21
Total Violations	302	50	0	0	252

Check ID	Category	Violations	Ignored	Compliance
MISRA12_1.1	Required	N/A	N/A	Compliant*
MISRA12_1.2	Advisory	N/A	N/A	Not Supported
MISRA12_1.3	Required	N/A	N/A	Not Supported
MISRA12_2.1	Required	0	0	Compliant

Control Coupling

Languages: Any
Targets: Project
Documents control coupling between functions: for each function, every function call it makes, classified as consumer, producer, or both.

Output mirrors Section 6.2 of the ETI / DO-178C control coupling report format. The table is filterable by Module and by function name.

Columns:

- **Module** – source file containing the function under analysis
- **Function Under Analysis** – the function making the call(s)
- **Consume** – called function whose return value/output is used by the function under analysis (N/A if not applicable)
- **Produce** – called function that the function under analysis drives/controls (N/A if not applicable)
- **Decision Point** – the condition/loop expression if the call appears inside a conditional or loop guard; otherwise N/A
- **SVCPs** – Software Verification and Certification Points, left blank for the analyst to fill in

A single call can appear as both Consume and Produce when the return value is used and the call also controls another component. Calls whose value is discarded are always Produce. Calls whose value is used are also Produce when the callee returns nothing or has visible side effects (writes to globals, statics, members, or through pointers).

Data Coupling

Languages: Any
Targets: Project
Documents data coupling between functions: for each function, every variable whose value is sourced from a call to another function.

Output mirrors Section 6.1 of the ETI / DO-178C data coupling report format. The table is filterable by Module and by function name.

Columns:

- **Module** – source file containing the consumer function
- **Destination (Consumer Function)** – the function consuming the data
- **Variable** – the variable that holds the data
- **Source** – the expression (containing the call) that sets the variable
- **Variable Data Type** – declared type of the variable
- **SVCPs** – Software Verification and Certification Points, left blank for the analyst to fill in

Calls appearing only in control statements (conditions and loop guards) are excluded – those belong in the Control Coupling report.

Data Dictionary

Languages: Any
Targets: Project

Generates a Data Dictionary: a multi-section reference document covering all functions, enums, constants, and variables in the project.

Output mirrors Appendix A of the ETI / DO-178C analysis report format. Each section is a separate page with a table filterable by source file and by name.

- **Functions** – name, description (from the doc comment or doxygen @brief), parameter types, and return type
- **Enums** – enum types with their enumerator values
- **Constants** – #define macros and const-qualified variables with their values
- **Variables** – type, documented valid range, initial value, scope (local/file/global), and accessor/mutator functions

Deviation Approval Records

Languages: Any
Targets: CodeCheck
One record block per deviation, approval first -- the shape a certification audit asks for.

DO-178C and DO-330 expect coding-standard deviations to exist as reviewable QA records: what was deviated from, where, why, and on whose authority. This report prints each deviation as its own block with **Approved By** and **Approval Date** leading, followed by the full record. Deviations nobody has approved are grouped first, as the open work.

The records are the ones suppressions already carry (see the Deviation Register); nothing needs to be re-entered for the audit.

Deviation Register

Languages: Any
Targets: CodeCheck
A register of every suppressed violation, with its justification and approval record.

When a coding standard requires that deviations be justified, reviewed and approved, this is the document that shows it. Each suppression is listed with the check it silences, where it is, who wrote it, and the structured record it carries.

- **Summary** How many deviations exist, how many are fully documented, and which checks and files they cluster in.
- **Register** The full table — one row per suppression, one column per field found in the records, sortable and filterable.
- **Gaps** The deviations that are not defensible yet: no justification at all, or missing a field that comparable deviations record. This is the work list.
- **By Check** Where deviations concentrate, and how completely each check's

deviations are documented.
- **By Field** Group the register by any field — Approved By, Severity, Owner — to review one owner's deviations at a time.

Deviations get their records from annotation templates offered in the Ignore Violation dialog, and read the same whether the suppression was stored as a comment in the code or as an annotation in the project. A suppression written without a template still appears here, counted as undocumented.

The Summary heads with the Deviation Completeness chart: the whole inspection split into what nobody deviated from, what is deviated from and defensible, and what is deviated from but not yet documented.

Deviation Risk Assessment

Languages: Any
Targets: CodeCheck
Every deviation with its recorded risk, and the ones missing one listed first.

Functional-safety standards (ISO 26262, IEC 61508, IEC 62304) require that a deviation from the coding guidelines documents its risk: what could go wrong because this code stays as it is. This report splits the inspection's deviations into the ones whose record carries a **Risk Assessment** and the ones whose record does not -- the second group is the assessor's work list, so it comes first.

The records are the ones suppressions already carry (see the Deviation Register).

Guideline Compliance Summary

Languages: Any
Targets: CodeCheck
One row per checked guideline, judged the way a compliance auditor judges it.

MISRA Compliance:2020 asks for a Guideline Compliance Summary: every guideline of the claimed standard, each marked **Compliant** (no violations), **Compliant with deviations** (every violation suppressed by a documented deviation), **Compliant with violations** (undeviated violations against an Advisory guideline only), or **Not compliant**. This report builds that table from the inspection's own configuration: one row per check, the check's name standing for the guideline, its category read from the check's *Category:* tag.

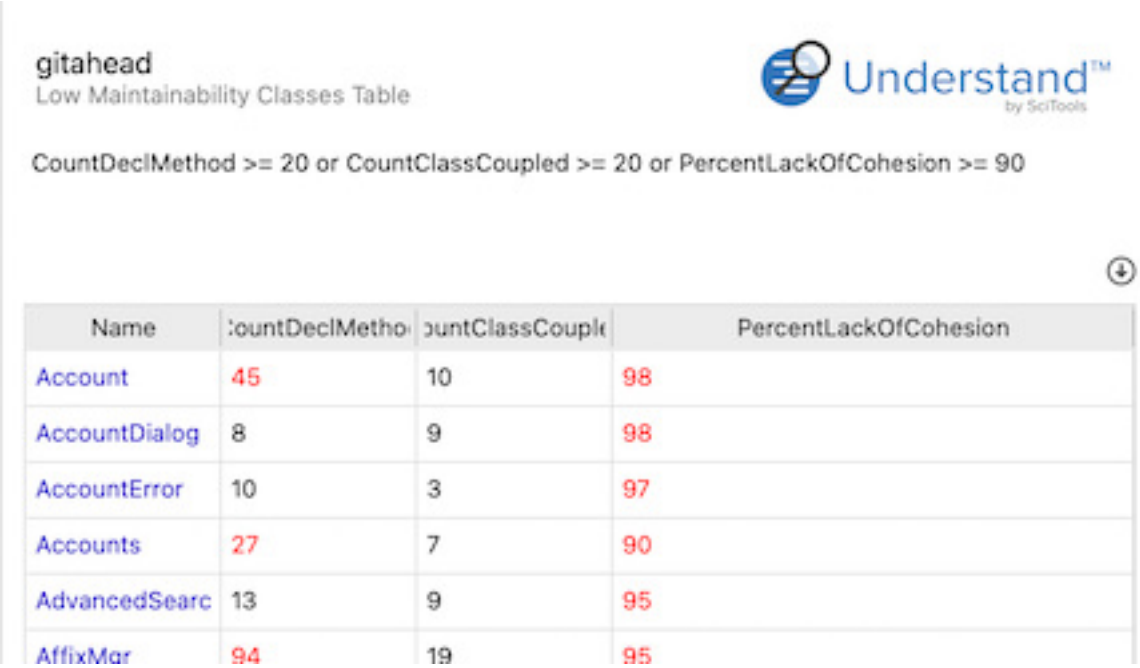
A suppression is a *deviation* only when it carries a documented record (see the Deviation Register); one that records nothing leaves its guideline non-compliant. Deviations against a **Mandatory** guideline are flagged: MISRA does not permit them at all.

Low Maintainability Classes Table

Languages: Basic, C#, C++, Java, Pascal, Python, Web
Targets: Project

Finds classes that exceed thresholds for specified metrics.

This report displays classes exceeding thresholds for CountDeclMethod, CountClassCoupled, and PercentLackOfCohesion, aiding in identifying potentially hard-to-maintain classes. Read more about it in the blog article ↗.



gitahead
Low Maintainability Classes Table

CountDeclMethod >= 20 or CountClassCoupled >= 20 or PercentLackOfCohesion >= 90

Name	:ountDeclMetho	ountClassCoupl	PercentLackOfCohesion
Account	45	10	98
AccountDialog	8	9	98
AccountError	10	3	97
Accounts	27	7	90
AdvancedSearc	13	9	95
AffixMgr	94	19	95

Low Maintainability Functions

Languages: Any
Targets: Project

Finds functions exceeding thresholds for Cyclomatic, CountLine, and MaxNesting metrics.

This report helps identify potential low-maintainability functions in your codebase. It flags functions based on user-defined thresholds for three key metrics: Cyclomatic Complexity, CountLine, and MaxNesting. By identifying and addressing these functions, you can improve the overall maintainability and quality of your code. Read more about it in the blog article ↗.

gitahead
Low Maintainability Functions



Cyclomatic >= 20 or CountLine >= 100 or MaxNesting >= 5

Name	Cyclomatic	CountLine	MaxNesting
Scintilla::UndoH	16	71	5
Application::Ap	6	128	3
Scintilla::CellBu	20	100	5
Scintilla::CellBu	33	169	5
Blowfish_initsta	1	278	0