Developing high-quality software is tough. ECLAIR is designed to help development, QA, and safety teams reach their quality goals

Coverage of IEC 60335-1

1 Introduction to IEC 60335-1:2020+AMD1:2025

IEC 60335-1:2020+AMD1:2025, "Household and similar electrical appliances — Safety — Part 1: General requirements," is consolidated Edition 6.1 of the general safety standard for electrical appliances intended for household and similar purposes [7, 8]. It is used together with the applicable Part 2 standard, which supplements or modifies the general requirements for a particular appliance type.

Software is increasingly important both in programmable electronic circuits that implement protective functions and in connected appliances that receive commands, data, or software updates through public networks. IEC 60335-1:2020+AMD1:2025 addresses these two concerns in two normative annexes:

Annex R — Software evaluation defines requirements for software used by programmable electronic circuits when software measures are needed to control specified fault/error conditions. It covers the software safety requirements, architecture, module design and coding, static analysis, validation, version management, and modification process. Amendment 1 also applies relevant Annex R requirements when a programmable electronic circuit is relied upon for the optical-radiation safeguards introduced in Clause 22.63 [8].

Annex U — Appliances intended for remote communication through public networks defines safety-oriented requirements intended to avoid unauthorized access and the effects of transmission failures where remote communication could impair compliance with IEC 60335-1. Its principal security goals are integrity and authenticity; it is not a general framework for confidentiality, privacy, or service availability [10].

Clause 22.62 establishes the applicability of Annex U. An applicability assessment is needed when remotely communicated software or data can affect Annex R measures, can affect means needed for compliance with Clauses 8 to 32, or can interfere with such software because software separation or partitioning is inadequate. Annex U is not applicable where all compliance measures are independent of software, where remote communication is send-only, or where the appliance only provides event-driven messages or push remote monitoring.

1.1 Role of ECLAIR in Supporting IEC 60335-1 Compliance

The ECLAIR Software Verification Platform provides objective and repeatable evidence for many Annex R activities and for the software-engineering aspects of Annex U. Its strongest contributions are:

- forward and backward traceability between requirements, code, and tests;
- formal specification and systematic checking of software architectural constraints, including layering, interfaces, and separation;
- static control-flow and data-flow analysis;
- verification of coding-standard compliance and controlled language subsets;
- enforcement of software-metric thresholds for modularity, complexity, testability, and maintainability;
- repeatable re-analysis after a modification, producing auditable regression evidence.

For Annex U, ECLAIR verifies properties of the software that implements the security and update mechanisms; it does not itself supply authentication, cryptography, secure boot, key management, a communications protocol, or product-level testing. The distinction is important: ECLAIR can provide strong source-code and architecture evidence without replacing the hardware, runtime, integration, and appliance tests required by the standard.

2 ECLAIR Coverage of IEC 60335-1 Annex R

Annex R requires a connected chain of evidence from software safety requirements to architecture, from architecture to module design and code, and from implementation back to the requirements through validation. The annex draws its systematic-fault-avoidance requirements from IEC 61508-3 and adapts them to the needs of household and similar appliances [9]. Amendment 1 makes software separation more explicit: it defines partitioning, software separation, safety-related software, non-safety-related software, and communication software, and adds Table R.8 and Figure R.1 [8].

In the following tables, **Direct** means that an ECLAIR analysis, report, or traceability result directly provides evidence for the identified activity. **Support** means that ECLAIR verifies part of the implementation or supplies supporting evidence, while additional design, review, or test evidence remains necessary.

As ECLAIR provides direct support for MISRA guidelines as well as guidelines from other coding standards, a reference for a guideline should be taken as a reference to the corresponding *ECLAIR service* as described in the *ECLAIR User's Manual*. For example, "MISRA C:2025 Directive 3.1" corresponds to the ECLAIR service `MC4.D3.1`, "MISRA C++:2023 Rule 9.4.1" corresponds to the ECLAIR service `MP2.9.4.1` and "BARR-C:2018 Rule 4.1.a" corresponds to the ECLAIR service `NC3.4.1.a`. For ECLAIR services that do not correspond to published coding standards, the service name is given in teletype font: for example, `B.INDEPENDENCE` is the name of an ECLAIR service that supports automatically enforcing software architectural constraints [1]. A complete definition of all ECLAIR services is contained in the *ECLAIR User's Manual* and, where applicable, in the corresponding coding standard documentation referenced therein.

2.1 MISRA C:2025

MISRA C:2025 [12] is the software development C subset developed by MISRA that is a de facto standard for safety-, life-, security-, and mission-critical embedded applications in many industries, including aerospace, railway, medical, telecommunications and others. MISRA C:2025, which allows

coding MISRA-compliant applications in subsets of C90, C99, C11 and C18, is supported, along with all previous versions of MISRA C, by the ECLAIR package called "MC".

2.2 MISRA C++:2023

MISRA C++:2023 [11] is the software development C++ subset developed by MISRA, which is a de facto standard for safety-, life-, and mission-critical embedded applications in many industries including aerospace, railway, medical, telecommunications and others. MISRA C++:2023 completely supersedes MISRA C++:2008 [13], the previous edition of the coding standard, which is still used by many legacy projects. MISRA C++:2023 and MISRA C++:2008 are supported by the ECLAIR package called "MP".

2.3 BARR-C:2018

The *Barr Group's Embedded C Coding Standard*, BARR-C:2018 [3], is, for coding standards used by the embedded system industry, second only in popularity to MISRA C. BARR-C:2018 guidelines include 64 guidelines dealing with language subsetting and project management as well as 79 guidelines concerning programming style. For projects in which a MISRA compliance requirement is not (yet) present, the adoption of BARR-C:2018 is a major improvement with respect to the situation where no coding standards and no static analysis is used. The adoption of the stylistic subset of BARR-C:2018 (79 out of 143 rules) can be part of complying with the MISRA requirement that a consistent programming style is adopted and systematically used as part of the software development process. Moreover, complying with BARR-C:2018, besides avoiding many dangerous bugs, entails compliance with a non-negligible subset of MISRA C:2012 [2]. ECLAIR support for BARR-C:2018 has no equals on the market: it is included in all ECLAIR packages, including the affordable package "B".

Table 1 — Derived mapping of Annex R activities to ECLAIR

Clause	**Objective**	**ECLAIR contribution**	**Level**
R.1, R.2	Ensure that software used to control the specified fault/error conditions does not impair compliance, and use an architecture appropriate to the required fault-control measures	ECLAIR verifies the implementation of software fault-control logic and architectural constraints. Selection and validation of the runtime architecture, including single- or dual-channel structures, require complementary system and test evidence	**Support**
R.2.2	Implement and inspect measures addressing memory diversity, monitoring, communication errors, safety-related data, timely fault detection, initialization, termination, and protection from alteration	Static control-flow and data-flow analyses, coding-rule checks, and project-specific ECLAIR services can verify relevant source-code properties. Hardware behavior and fault-injection or dynamic tests remain outside static analysis	**Support**
R.3.2.1	Specify safety functions and timing, controlled faults, hardware/software interfaces, safety/non-safety interfaces, and the compiler and linker configuration	`B.REQMAN` supports traceability from safety requirements and interfaces to implementation and tests. ECLAIR analysis configurations also record the compiler model and relevant implementation-defined aspects	**Support**

continued

Clause	Objective	ECLAIR contribution	Level
R.3.2.2.1, Table R.8, and Figure R.1	Specify the architecture, including fault-control measures, module partitioning and allocation to safety-related, communication, and non-safety-related functions, exclusive management of safety-related data by safety-related code, control flow, interrupt handling, data flow and access, data storage, and timing dependencies	`B.INDEPENDENCE` formalizes and checks architectural policies. ECLAIR control-flow and data-flow analyses reveal implemented interactions between modules and data. Evidence for runtime resource isolation and any hardware-, operating-system-, or hypervisor-enforced separation remains complementary	**Direct**
R.3.2.2.2	Validate the architecture specification against the software safety requirements by static analysis	`B.REQMAN` connects requirements to implementation evidence; `B.INDEPENDENCE`, control-flow analysis, and data-flow analysis systematically check the implemented design. Review or other evidence is also needed to establish the semantic relationship between specifications	**Support**
R.3.2.3.1–R.3.2.3.2	Refine the architecture into traceable modules with defined functions, interfaces, and data, and use structured code	MISRA C, MISRA C++, BARR-C:2018, ECLAIR-specific services, and software metrics enforce disciplined module design, interfaces, language use, and structural complexity	**Direct**
R.3.2.3.3	Validate code against the module specification and the module specification against the architecture by static analysis	This is a core ECLAIR use case: source-code analysis, requirements traceability, architectural-constraint checking, and documented reports provide repeatable implementation evidence. The semantic relationship between the module and architecture specifications also requires review or complementary evidence	**Support**
R.3.3 and Table R.7	Validate the software against the software safety requirements using representative normal, anticipated, and undesired conditions, with documented test cases, data, and results	`B.REQMAN` traces requirements to tests and code; ECLAIR metrics help assess testability and analysis reports complement validation. Simulation and test execution remain separate activities	**Support**
R.3.4	Uniquely identify software versions and control modifications through impact analysis, specified verification and validation, re-verification, and documentation	Versioned ECLAIR configurations and reports provide reproducible baselines. `B.REQMAN` supports impact analysis and traceability, while repeated analysis supplies regression evidence for changed and affected code	**Support**

2.4 HIS and Other Source Code Metrics

Source code metrics are recognized by many software process standards (and from MISRA) as providing an objective foundation to efficient project and quality management. One well known set of metrics has been defined by HIS (Herstellerinitiative Software, an interest group set up by Audi, BMW, Daimler, Porsche and Volkswagen).

The *HIS source code metrics* [4], while well established, include some metrics that are obsolete and miss others that are required or recommended by software process standards, such as those that allow

estimating function coupling. For this reason, ECLAIR supplements HIS source code metrics with numerous other metrics that allow software quality to be assessed in terms of complexity, testability, readability, maintainability and so forth. Keeping track of these metrics also provides an effective and objective method to assess the quality of the software development process. The full set of metrics is available in all ECLAIR packages.

2.5 Selected Annex R Module-Design and Coding Measures

Tables R.5 and R.6 identify techniques and measures for modular, structured, and analyzable software. Table 2 summarizes the most direct ECLAIR contributions without reproducing the normative tables.

Table 2 — ECLAIR support for selected Annex R module-design and coding measures

Technique or measure	**ECLAIR support**	**ECLAIR**
Coding standard	Complete, automated support for MISRA C, MISRA C++, and BARR-C:2018, including controlled language subsets and project-specific policies	✓
No dynamic objects or variables	MISRA services enforce restrictions on dynamic allocation. Annex R permits an exception only where the compiler guarantees allocation before runtime or inserts checks for correct online allocation; evidence for those compiler guarantees is complementary to ECLAIR analysis	✓
Limited pointer use	MISRA services and the ECLAIR-specific services `B.PTRDECL` and `B.PTRUSE` provide fine-grained control of pointer declarations and operations	✓
Limited recursion	MISRA rules prohibit recursion where required; call-graph metrics can detect cycles across the analyzed program	✓
Structured control flow	Coding-rule checks constrain unconditional jumps and other unstructured constructs; `HIS.GOTO` can impose a measurable project threshold	✓
Limited module size and complexity	ECLAIR metrics measure size, cyclomatic complexity, acyclic paths, coupling, and related quality properties against project thresholds	✓
Information hiding and fully defined interfaces	MISRA rules support encapsulation and interface consistency; `B.INDEPENDENCE` checks that dependencies and accesses respect the specified interfaces	✓
Static control-flow, data-flow, and run-time error analysis	ECLAIR reasons about execution paths, pointers, values, dead stores, and classes of run-time errors without executing the program	✓

2.6 Architecture, Partitioning, and Independence

Annex R makes software architecture an explicit verification object. The architecture specification addresses allocation of modules to safety-related, communication, and non-safety-related functions, hardware/software interactions, call structure, interrupts, data flow and access restrictions, data storage, and time dependencies. It is then validated against the software safety requirements by static analysis.

Table R.8 states the principles for software partitioning and Figure R.1 illustrates software-separation arrangements. Separation can be physical or virtual; where partitions share a processor, the evidence must address the software architecture and the mechanisms that protect memory, execution, and interfaces. This makes source-visible data ownership and cross-partition dependencies especially important verification targets.

The *ECLAIR Independence Checker* (service `B.INDEPENDENCE`) provides direct support for this task. It can detect and check interactions between user-defined software elements that occur through:

- reads and writes of shared memory;
- function calls and the passing or returning of data;
- header inclusion and macro expansion;
- dependencies that bypass an intended interface or architectural layer.

The project defines components, private and shared program elements, and the interactions that are permitted between them. ECLAIR reports each implemented interaction that violates that policy. This turns architecture review from a one-time manual exercise into a repeatable check that can be run on every relevant software baseline.

3 ECLAIR Coverage of IEC 60335-1 Annex U

Annex U is concerned with appliance safety when remote communication through a public network could impair compliance with the standard. It combines requirements for software separation and partitioning, robust message handling, authorization and authentication, data integrity, update control, and safe behavior when communication is unavailable.

The strongest direct ECLAIR contribution is the software-separation evidence required by U.3.1 and by the Clause 22.62 applicability analysis. For the remaining construction requirements, ECLAIR can verify the software implementation and its architecture, but cannot replace evaluation of the protocol, cryptographic mechanisms, complete update chain, or appliance behavior. IEC TC 61 guidance aligned with Amendment 1 provides additional interpretation of these requirements [10].

Table 3 — Derived mapping of Annex U activities to ECLAIR

Clause	Objective	ECLAIR contribution	Level
U.2.1	Identify the installed software version and provide update instructions	ECLAIR configurations, reports, and requirements traceability support software-version evidence, but version display and user instructions are separate product deliverables; the version can be displayed by an app or another device	**Support**
U.3.1	Provide correct software separation between communication software and software needed for compliance with the standard, applying the Annex R partitioning principles	`B.INDEPENDENCE` checks separation policies, allowed interfaces, data access, calls, and prohibited cross-partition dependencies. Runtime and platform isolation mechanisms require complementary evidence	**Direct**
U.3.2	Protect communication data integrity and detect or handle corruption, addressing errors, timing or sequence errors, repetition, interruption, malformed messages, and out-of-range information; select appropriate measures from Table U.1, apply measures for the Table R.1 fault/error conditions, and evaluate the architecture and software under R.3.2.2 and R.3.3	Static analysis verifies message-validation and error-handling code, control and data flow, range checks, initialization, and relevant coding rules, and supports the R.3.2.2 architecture evaluation. R.3.3 validation, protocol tests, and fault-injection tests remain necessary	**Support**

continued

Clause	Objective	ECLAIR contribution	Level
U.3.3	Avoid hazards caused by messages received from multiple sources, simultaneously or sequentially	ECLAIR can analyze the implementing dispatch, state, validation, and error-handling logic. Concurrency, timing, and system behavior also need dynamic evaluation	**Support**
U.3.4	Enable remote communication only after authorization based on cryptographic authentication of both parties	ECLAIR can verify secure coding, control flow, data flow, and architectural isolation of authentication and authorization code; it does not provide the authentication mechanism	**Support**
U.3.5 and Table U.1	Prevent unauthorized access and detect transmission faults/errors	ECLAIR verifies the implementation of selected countermeasures from the updated, non-exhaustive table and their integration with the rest of the software. Effectiveness of the complete communications system needs separate security and fault testing	**Support**
U.3.6	Ensure that safe operation does not depend on remote communication	`B.INDEPENDENCE` can demonstrate prohibited dependencies from safety-related software to communication modules and network state. Appliance tests with communication disabled remain necessary	**Support**
U.3.7	Apply cryptographic techniques for communication data integrity after authorization; the techniques are part of the appliance or accessories, are applied before transmission, and do not rely on the router or similar transmission device	ECLAIR can detect implementation defects and enforce boundaries around cryptographic code. Algorithm selection, key management, cryptographic strength, and protocol assurance require specialist evidence	**Support**
U.3.8	Verify manufacturer-provided updates for transmission corruption and appliance compatibility before installation, with fault-control measures in the checking software	Requirements traceability, static analysis, architecture checking, and versioned reports support verification of the update-checking software. Provenance, signatures, secure boot, rollback, and end-to-end update tests require complementary measures	**Support**
U.3.9	Obtain permission from the person responsible for each installation, or for an enabled automatic-update mode	ECLAIR can verify relevant implementation code, but the permission model, user interaction, and functional behavior are primarily product-level concerns	**Support**
U.3.10	Preserve compliance with the standard during and after software installation	Architecture checks and repeated static analysis provide regression evidence for changed and affected software. Safe-state behavior and post-installation compliance require relevant product testing	**Support**

3.1 The Architectural Bridge Between Annexes R and U

Annexes R and U meet at the software architecture. Annex R requires partitioning, controlled data access, explicit interfaces, and static validation of architecture against the safety requirements; Table R.8 and Figure R.1 make the partitioning and separation expectations explicit. Annex U requires communication software to be separate from software needed for product safety and uses inadequate separation as one of the reasons why remotely changed software can enter its scope.

An ECLAIR-based evidence chain can therefore be organized as follows:

1. `B.REQMAN` links software safety and remote-communication requirements to architecture, code, and tests.
2. `B.INDEPENDENCE` formalizes the allowed dependencies between communication, non-safety-related, and safety-related components.
3. MISRA and project-specific services enforce safe and structured code, while metrics keep modules and interfaces reviewable.
4. ECLAIR control-flow, data-flow, and run-time error analyses verify the implementation and generate auditable reports.
5. The same configured analyses are repeated after an update or other modification to provide regression evidence.

Typical project policies include prohibiting communication modules from managing safety-related data; permitting validated information to cross into the safety-related partition only through reviewed interfaces; preventing network callbacks from bypassing the command-validation layer; constraining calls from safety-related software into communication services; and limiting shared state to explicitly reviewed objects.

4 Confidence in ECLAIR for IEC 60335-1 Projects

The software-development requirements in Annex R are extracted from IEC 61508-3 and adapted to IEC 60335-1. IEC Technical Committee 61 guidance also identifies appropriate tool qualification and increased confidence from use as relevant considerations for software design, verification, and maintenance tools [9].

ECLAIR is a class T2 off-line support tool under IEC 61508 [6, Clause 3.2.11] and meets the IEC 61508 Part 3 requirements applicable to such tools [5, Clause 7.4.4]. TÜV SÜD has awarded BUGSENG the "Software Tool for Safety Related Development" Certificate no. Z10 116151 0001 Rev. 01, attesting that ECLAIR is suitable for safety-related development according to IEC 61508:2010 for any SIL. The ECLAIR tool-confidence argument in compliance with IEC 60335-1 can thus be based on the ECLAIR FuSa Pack.

5 The Bigger Picture

ECLAIR is very flexible and highly configurable: it supports all kinds of software development workflows and environments.

ECLAIR is fit for use in mission- and safety-critical software projects: it has been designed from the outset to exclude configuration errors that would undermine the significance of the obtained results.

ECLAIR is developed in a rigorous way and carefully checked with extensive internal test suites (tens of thousands of test cases) and industry-standard validation suites.

ECLAIR is based on solid scientific research results and on the best practices of software development.

ECLAIR's unique features and BUGSENG's strong commitment to the customer, allow for a smooth transition to ECLAIR from any other tool.

BUGSENG's quality system has been certified by TÜV Italia (TÜV SÜD Group) to comply with the requirements of UNI EN ISO 9001:2015 for the "Design, development, maintenance and support of tools for software verification and validation" (IAF 33).

BUGSENG is an Arm's Functional Safety Partner, and is thus recognized as a partner who can reliably support their customers with industry leading functional safety products and services.

For More Information

BUGSENG srl
Via Fiorentina 214/C
I-56121 Pisa, Italy
Email: info@bugseng.com
Web: http://bugseng.com

bugseng

**no shortcuts,
no compromises,
no excuses:
software verification done right**

References

[1] R. Bagnara, A. Bagnara, and P. M. Hill. Formal verification of software architectural constraints. In DESIGN&ELEKTRONIK, editor, *embedded world Conference 2023 — Proceedings*, pages 271–279, Nuremberg, Germany, 2023. WEKA FACHMEDIEN, Richard-Reitzner-Allee 2, 85540 Haar, Germany.

[2] R. Bagnara, M. Barr, and P. M. Hill. BARR-C:2018 and MISRA C:2012 (with Amendment 2): Synergy between the two most widely used C coding standards. In DESIGN&ELEKTRONIK, editor, *embedded world Conference 2021 DIGITAL — Proceedings*, pages 378–391, Nuremberg, Germany, 2021. WEKA FACHMEDIEN, Richard-Reitzner-Allee 2, 85540 Haar, Germany.

[3] M. Barr. *BARR-C:2018 — Embedded C Coding Standard*. Barr Group, www.barrgroup.com, 2018.

[4] H. Kuder et al. HIS source code metrics. Technical Report HIS-SC-Metriken.1.3.1-e, Herstellerinitiative Software, April 2008. Version 1.3.1.

[5] IEC. *IEC 61508-3:2010: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems — Part 3: Software Requirements*. IEC, Geneva, Switzerland, April 2010.

[6] IEC. *IEC 61508-4:2010: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems — Part 4: Definitions and Abbreviations*. IEC, Geneva, Switzerland, April 2010.

[7] IEC. *IEC 60335-1:2020: Household and Similar Electrical Appliances — Safety — Part 1: General Requirements*. IEC, Geneva, Switzerland, September 2020.

[8] IEC. *IEC 60335-1:2020/AMD1:2025: Household and Similar Electrical Appliances — Safety — Part 1: General Requirements — Amendment 1*. IEC, Geneva, Switzerland, October 2025.

[9] IEC Technical Committee 61. Revised guidance document concerning functional safety of appliances using programmable electronic circuits. Technical Report 61/6434/INF, IEC, Geneva, Switzerland, October 2021. Available at https://assets.iec.ch/public/tc61/61-6434-INF.pdf.

[10] IEC Technical Committee 61. Guidance document concerning the safety of appliances communicating remotely through public networks. Technical Report 61/6124B/INF, IEC, Geneva, Switzerland, May 2026. Available at https://assets.iec.ch/public/tc61/61_6124B_INF.pdf.

[11] MISRA. *MISRA C++:2023 — Guidelines for the use of C++17 in critical systems*. The MISRA Consortium Limited, Norwich, Norfolk, NR3 1RU, UK, October 2023.

[12] MISRA. *MISRA C:2025 — Guidelines for the use of the C language in critical systems*. The MISRA Consortium Limited, Norwich, Norfolk, NR3 1RU, UK, March 2025.

[13] Motor Industry Software Reliability Association. *MISRA C++:2008 — Guidelines for the use of the C++ language in critical systems*. MIRA Limited, Nuneaton, Warwickshire CV10 0TU, UK, June 2008.