AdaCore DIGEST

OCTOBER 2026

AdaCore

DIGEST

Newsletter 10/26

In This Issue

The main theme of this issue of the AdaCore Digest is the new content in the 27.0 release, so it is a packed issue with plenty of useful technical detail to dig into.

We also cover the release of GNAT Foundry: Intersection, an open-source demonstrator built by AdaCore on how you can build deterministic trust in artifacts built by Artificial Intelligence. Related to this topic, we have a section on how lifting Ada to SPARK Silver through the use of AI helps find and fix defects in existing Ada based projects. The section on the Flare extensions to Ada gives you a good idea of some of the newer additions to Ada that are in the upcoming compiler versions. The Ada SPARK office hours meeting is a great place to discuss your experiences related to Flare, SPARK and AI.

AdaCore is also working with the Rust open source projects to implement MC/DC support into the open source Rust compiler. This is a necessity for projects that want to use Rust in projects that require a high degree of functional safety such as DO-178C DALA, or ISO 26262 ASIL D. We have a topic to update you on our progress in this area.

We hope you enjoy this issue of the AdaCore Digest, and feel free to reach out with any suggestions for future content.

GNAT Foundry: Intersection

Many software development projects are trying to figure out how to trust code generated by Artificial Intelligence. The reason is logical: AI is reducing the cost of software development; projects can do more with less effort in shorter timeframes. Projects can use that time to deliver greater value to end customers or to bring a full-featured project to market earlier.

The problem, of course, is that AI is inherently non-deterministic and hence not trustworthy. However, so are human engineers. They make mistakes, have off days, and sometimes get distracted. That is why high-integrity software development projects follow a meticulous process to ensure that mistakes are caught and that off-days do not impact software quality.

AdaCore's vision here is that Artificial Intelligence is fundamentally untrustworthy, but that we can use the same process we use for humans to build trust in AI-created artifacts. We can use deterministic

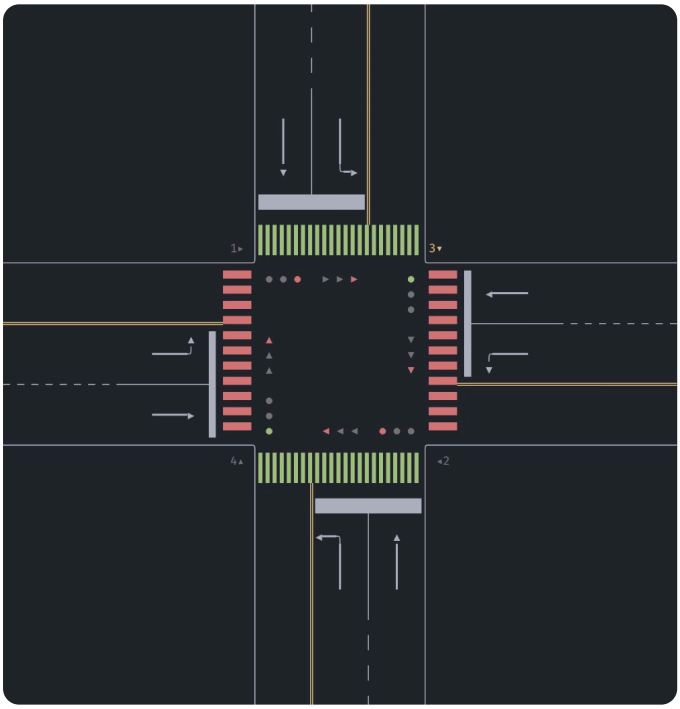
tools to increase trust in artifacts created by non-deterministic creators, humans, or AI.

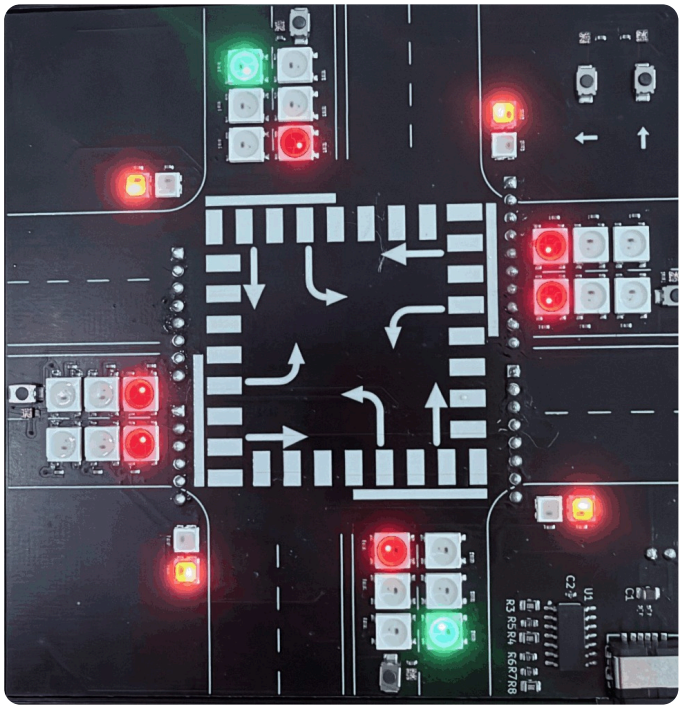
AdaCore's tools are engineered to build that trust:

- Ada and SPARK are designed to maximize static checks using formal methods
- GNATprove validates the properties expressed by formal methods at build time
- CodeSonar® offers static analysis
- GNAT DAS offers testing and coverage analysis
- Ada's run-time checks provide a safety net during execution

Our vision is that AI can reduce the cost of developing high-integrity software, enabling more projects to adopt these practices. More high-integrity software is a net win for society. Imagine an IoT device built with a higher degree of security, imagine more powerful, more secure medical devices, robots, you name it.

We built a demonstrator project of a traffic light control system in which we laid out this vision and made it available to the software development community. The traffic light system can run natively on a workstation or on an embedded MCU. You can find the GitHub repository here, and we also wrote a blog post explaining how to explore it. There is also a short introductory video.





We encourage you to review the content and reach out to your AdaCore account team to discuss. We value your feedback. Alternatively, feel free to join us for the Ada SPARK Office Hours (more info later in this Digest), where we discuss these topics.

27.0 Release

As always, the .0 release includes many new features, as well as frontend and backend updates. The list below is a short overview of the content. Please reach out to your AdaCore contact if you would

like more details on any of the topics. Our sales engineering and product management teams thrive on customer feedback and interactions.

We will organize a webinar with more content on these features in Q1-2027, so keep on the lookout for that. You can always see current and future roadmaps on the support part of our website.

GNAT Pro

- Introduction of Alire Pro, the Ada LIbrary REpository as a supported package from AdaCore, including a selection of libraries
- Transition to GCC-15 and LLVM 21 to remain in sync with those ecosystems
- VxWorks Support
  - GNAT Pro now supports VxWorks 25.09
  - GNATcoll support for VxWorks 7 on all supported architectures
- Ability to compile Ada to 64-bit Arm Android targets
- GNATpolyglot enhancements to Java language support
- New LLVM Compilation Support
  - Ability to compile C and Ada using LLVM for 32-bit Linux and 32-bit and 64-bit Windows targets (x86)
  - Support for shared and static position-independent Ada runtimes with the LLVM back-end
- GNAT Pro for Rust
  - No-std support for 32-bit Arm bare-metal, including libtest support
  - Support for Rust version 1.93.0
  - Improved support for debugging Rust on VxWorks targets using vxlink and vxgdb
  - Support for gprbuild2
  - MC/DC support
- Ada language extensions
- GNATformat adds support for upper/lower casing and configuration options for layout
- GPR2 library released into production (libgpr2)
- GtkAda upgrade to Gtk4

SPARK Pro

Language evolution

- Allow functions with side-effects in more locations
- Precise global and parameter modes by proof
- Precise support for overlays
- Refer to loop index in for .. of loops
- Refer to values at labels
- Unchecked Union Types

User Experience

- Access type conversions in the SPARKlib
- Feedback on why provers stopped working on unproven checks
- Show if the proof comes from a cache
- Tree container library in the SPARKlib

CodeSonar

GNAT SAS is now CodeSonar for Ada

We are excited to introduce CodeSonar for Ada, the evolution of our static analysis suite. In addition to the powerful capabilities of GNAT SAS that you rely on, CodeSonar for Ada adds the hub, a web-based user interface for exploring, managing, and storing static analysis findings. Best of all, if you are a GNAT SAS user today, you don't have to do anything special to get it. You are entitled to CodeSonar for Ada today, and the next time you visit GNAT Tracker, you should see a brand new download. For more information or to see a demo, please reach out to your account team.

CodeSonar for Ada

- Even well-written code in a memory-safe language can have vulnerabilities. As part of our plan to provide checkers for the CWE Top-25 Weaknesses, we are adding a checker for CWE-502: Deserialization of Untrusted Data. We will also be adding checkers for CWE-80, CWE-113, CWE-117, CWE-393, CWE-400, CWE-402, and CWE-617.
- Extended API taint coverage
- Interprocedurality for Type State
- Quick fixes for GNATcheck rules, either interactively in VSCode or on the CLI
- GNATcheck findings can be reported in SARIF for better tool interoperability
- CodeSonar for Ada also supports analyzing Python, Rust, Go, JavaScript, and TypeScript source code.
- Static typing support in LKQL

CodeSonar for C/C++

- SOFTUNE compiler model modernization
- Improved support for the newest C and C++ language features
- Increased MISRA C++:2023 support

GNAT DAS

GNATcoverage

- Origin reporting to, for example, filter on a specific test
- Expanded build system support for non-gpr cross targets and Windows hosts
- Incremental instrumentation, improving build times
- Support for finer-grained exemptions when full MC/DC is not possible

GNATfuzz

- Support for fuzzing on x86_64 Windows
- Atomic commands for greater modularity
- Stabilization of the Fuzz-Everything workflow
- Hot-spot workflow to automatically validate SARIF reports using fuzzing
- Extension of the Adaptive Fuzzing Engine to on-target hardware platforms not supported by traditional fuzzers (beta feature, not for production)
- Support for automatic fuzzing of C-based applications (beta feature, not for production)

GNATtest

- jUnit report format support for integration into CI/CD platforms
- Support for stubbing of subprograms from another language, such as thin-bindings to C
- Support for automatic test case generation for class-wide types and access types

GNAT Studio and VS Code

- GNATformat improvements in GNAT Studio and VS Code
- Additional capabilities (scenario view, project navigation, and GNATcoverage) in VS Code
- SARIF Quick Fixes in GNAT Studio and VS Code
- Coverage percentages visible in GNAT Studio
- Alire and GPR editing improvements in GNAT Studio and VS Code
- GNATdoc improvements for Markdown fenced code blocks

Ada SPARK Office Hours

The Ada SPARK Office Hours is an event that the community team at AdaCore organizes every two weeks on Fridays from 10.00-11.00 Eastern Time. It is an opportunity for people to call in and discuss matters related to Ada and SPARK that interest them. Attendees have been Ada enthusiasts from academia, open-source projects, and commercial companies that use Ada to build the systems that run the world.

Lots of different topics have passed the review. A big topic of interest to the audience has been artificial intelligence and how to use it with both commercial and open-weight models to write Ada code and/or to lift existing Ada code to SPARK.

Another popular topic has been how to write platform-independent Ada that can run on a series of different boards against a flexible hardware abstraction layer.

News such as the new Ada extensions that AdaCore is presenting, as well as the GNAT Foundry - Intersection work, has also been discussed.

Topics for the upcoming office hours are usually posted on the Ada Forum in the Events category a couple of days before the event. Recordings are also published there a couple of days after the event.

The office hours are open to anyone and are a great way to keep up with what is going on in the community. The calendar invite is available on the AdaCore website.

Using AI to Lift Ada to SPARK Silver

We have written before (for example: Migrating C to Ada SPARK and Mars Rover GNC) in both the AdaCore Digest and on the AdaCore blog about how the progression in the capabilities of frontier artificial intelligence models over the past 9 months has been nothing short of astonishing. One of the use cases where the frontier models are showing their strength is lifting existing Ada code to SPARK Silver to statically prove the absence of runtime errors. AI is really lowering the barrier of entry to formal methods, which is a fantastic benefit for people writing Ada code.

As with any work with AI models, you need some sort of measurement of 'correctness'. People often refer to this as 'the oracle', which is a fitting name. The presence of an oracle helps AI modify a code base without breaking anything. A comprehensive unit testing suite is perfect for this.

Over the past couple of months, we have taken a number of open-source projects and used frontier models to lift the Ada code to SPARK, with the exception of the boundary where the code interacts with hardware or the OS. The results have been really encouraging.

Firstly, in all cases, the AI model was able to lift the model to SPARK without invalidating the unit tests. Secondly, in all cases, it found defects in the original source code, typically in the handling of corner cases around array boundaries or input validation.

One example is described in this GitHub Issue:BZip2.Decoding.RLE_1 latent bug. A value from a header field was used without a bounds check. This would have Constraint_Error instead of a data_error. Ada is still protecting the programmer through a runtime error; SPARK was able to find the problem statically.

Another example was in the AdaCL crate, specifically in the newly added adacl_serial module. This merge request documents a set of fixes, discovered through lifting to SPARK, that have since been integrated. There have been several other examples in both open source and commercial projects.

A useful tool in these workflows is to first install the gnatprove skill from https://github.com/AdaCore/skills and then ask the AI model to "use the gnatprove skill to lift the code base in <dir> to SPARK Silver without breaking any of the unit tests defined in the projects". This will get the model to explore the codebase and the unit tests, then fire off a series of subagents to lift the leaf nodes to SPARK, and, through that, slowly lift the entire codebase.

The result truly is interesting. If you are building high-integrity software in Ada and have access to frontier models, you really should try this workflow and see what defects you find. If you are a customer, reach out, and we can provide you with access to SPARK Pro, or you can use the community version to get started. When you do, please reach out and let us know what you learn.

Rust: MC/DC Contributions Merged to Upstream

Memory safety is rapidly becoming a requirement for projects developing high-integrity software. The Rust programming language is one of the languages that provides memory safety, and it is increasingly used in safety-critical contexts. Several projects have been delivered against industrial functional safety standards, such as IEC 61508, and ISO 26262 is increasingly used in these contexts as well, all at medium safety levels.

AdaCore has been working to prepare Rust for use in projects that must comply with Functional Safety regulations. We collaborated on the first-ever qualification of the Rust compiler against ISO 26262, worked on the Ferrocene Language Specification, and are continuing to collaborate with the Safety Critical Rust Consortium to advance the state of functional safety in the Rust ecosystem.

Our goal is certification to the highest levels of DO-178C, ISO 26262, and related standards, and we believe this is ultimately achievable. But there is one capability Rust still lacks: support for code coverage up to Modified Condition/Decision Coverage (MC/DC). While MC/DC support is not required by all functional safety standards, it is a recommended capability for the highest levels of certification.

The AdaCore team recently delivered support for MC/DC as a pull request to the Rust compiler open-source project. You can follow the progress in this GitHub issue, we are still working with the maintainer teams to get the changes merged into the upstream repositories. In languages like C, C++, and even Ada, MC/DC is a well-understood capability. With a language like Rust with pattern matching, this is not as clear-cut. The initial support for MC/DC does not support pattern matching yet, we hope to work with users and assessors to understand what is feasible in future versions.

AdaCore is committed to delivering this capability to the open-source ecosystem and then enabling it as part of our GNAT Dynamic Analysis Suite (DAS), so projects can depend on it in their functional safety projects using Rust.

This, combined with our ongoing effort to qualify parts of the Rust core library to DO-178C DAL-A, will enable the next step toward using Rust for projects that require higher levels of safety certification.

Overall, the ability to use Rust in projects to the highest levels of certification will give AdaCore's customers a choice of languages for their next-generation functional safety project. They can choose the best language for the job and even use GNATpolyglot to create idiomatic bindings to combine them into larger systems.

Flare: The Extension of Ada

At AdaCore, we spend a lot of time listening to our customers and the community about topics close to their hearts related to the progression of Ada and SPARK as languages. We have been collecting this feedback and steadily working to act on it with a goal of making it easier for humans and machines to develop software in Ada. We have been using the name 'Flare' for this Ada extension initiative, and over the next while we will highlight the various improvements here in the AdaCore

Digest, on the blog, in our Ada SPARK Office Hours, and in various online and in-person presentations. Flare is a multi-year endeavor, and we will release Flare 0.1 first. The extensions will be implemented in GNAT Pro and in compilers available to the community at large, with capabilities being integrated over the coming months and years. We hope that you will have a chance to explore them, and please get back to us with your feedback.

Backward Compatibility

The Flare extensions are designed to coexist with all versions of Ada and can be enabled with pragmas or command-line switches. The language choice unit is the compilation unit, which is either a Flare unit or an Ada unit. A unit can be 'pragma Flare' or compiled with -gnat_flare, which enables the pedantic version of Flare.

An extension is either additive, adding new capabilities without altering the meaning of existing Ada constructs, or changing. The latter alters syntax or semantics in a way that cannot be retrofitted onto Ada code without risk. 'pragma Flare_Extensions_0_1' or switch -gnat_flare_extensions enables just the additive changes and provides a gentle on-ramp.

Units can be mixed; a Flare unit can depend on an Ada unit, but not the other way around. This enables teams to use Flare in newer code, without having to rewrite existing code. Flare is also designed to fully work with SPARK, although capabilities are implemented first in the compiler, then in SPARK.

Rationale

We are working on a detailed rationale document that explains how the new features work, and we will publish that soon. As a sneak preview, we will provide a high-level summary of the key points on how Flare is evolving object-oriented programming in Ada.

Object-Oriented Programming

Ada has had object orientation since Ada 95, supporting core OOP paradigms, including polymorphism and type inheritance. Thirty years later, we now have experience showing that there are a number of shortcomings in OO in Ada that make it hard to use, especially for programmers coming from other languages.

The class record

Flare 0.1 proposes a number of modifications to Object Oriented Programming in Ada, driven by a new construction, the 'class record'. We chose to take this route rather than retrofit the new concepts into tagged types. There is a capability to extend existing tagged types with some of the new Flare OOP capabilities; more on this in a later article.

Note that the specific syntax of the class record is still under implementation and will be delivered early in the GNAT 28 cycle. However, most of the capability added to them, such as the ability to declare primitives within scope or constructors / destructors has also been made available for

traditional tagged types.

With Flare, primitives (member functions in other languages) are declared at the same lexical level as the type they are a primitive of, and can be either public or private.

```
Flare
type Root is class record
       C1 : Integer; -- A component

       procedure Prim (Self : Root); -- A primitive
end Root;
```

A class record can be derived from another with 'is new ... with class record':

```
Flare
type Child is new Root with class record
      C2 : Integer;

      overriding procedure Prim (Self : Child);

      procedure New_Prim (Self : Child);
end Child;
```

Only single inheritance is allowed. Interfaces remain available for multiple-inheritance-of-specification patterns. A class record may implement one or more interfaces in addition to extending a single parent class record.

The main benefit of that notation is to clarify the relationship between primitives and their related types. Primitives are now structurally "part of" their type - a bit like Ada protected objects.

Dispatching

One of the most common mistakes when developing Ada code is inadvertently making a static call to a primitive that the author intended to be dispatching. This is generally unsafe - primitives are overridden specifically to provide implementation consistent with the specific type they operate on, calling a parent version is usually a mistake and may end in inconsistent objects (except in very specific cases - e.g. calling the parent primitive from the overridden one). Flare fixes this gap by making all calls to primitives of class types dispatching by default. When the object's actual tag can be determined statically, the compiler is free to turn the dispatching call back into a static one for optimization - the semantic remains calling the primitive intended for the type.

Flare also introduces a 'Super' view of an object in case a static call is needed. For Self being of some kind of Child object, Self'Super.Prim() would call the Prim primitive in the Root class.

Constructors

The concept of a constructor has been a pattern missing in Ada. In most OOP languages, this forms an important part of the design. Flare introduces constructors through a special attribute, named

'Constructor', of the form <type>'Constructor, whose first parameter is of the class type in mode 'in out', optionally followed by further parameters. If you are familiar with constructs for languages such as C++ or Java, you will find a lot of commonalities.

```
Flare
type Root is class record
       procedure Root'Constructor (Self : in out Root);
       procedure Root'Constructor (Self : in out Root; Some_Parameter : Integer);
end Root;
```

Parameterless constructors are called implicitly at object creation. Constructors with parameters must be called explicitly through the <type>'Make attribute.

A class record always has at least one constructor. If none is declared, a default parameterless constructor is generated, which means that a type that offers only parameterized constructors cannot be created without 'Make.

There are many more details about constructors related to inheritance, component initialization, copy constructors, and, of course, destructors.

There are many more details of course related to object oriented programming, but as we mentioned, we will provide more details in the upcoming while.

Recent Blog Posts

Our blog at adacore.com has a steady stream of interesting content. Here is a selection of recent posts:

If you are interested in reading more about formal methods and SPARK:

- memcp: token-efficient, multi-project, multi-surface external memory for AI agents written in SPARK Silver https://www.adacore.com/blog/memcp-token-efficient-multi-project-multi-surface-external-memory-for-ai-agents-written-in-spark-silver
- Multiple Levels of Models and Refinement https://www.adacore.com/blog/formally-verified-hashed-sets-in-ada-spark-2
- From Raw Arrays to Typed Ownership: A Layered Allocator in SPARK https://www.adacore.com/blog/from-raw-arrays-to-typed-ownership-a-layered-allocator-in-spark

The latest Rust developments are outlined here

- https://www.adacore.com/blog/rust-for-do-178c-and-iso-26262

Have you seen the resources section on our website? From there, you can read not only the Blogs, but also Case Studies and Papers. We also have a dedicated newsroom where our most recent press releases and editorials are available.

Events

We have a full events listing on the AdaCore website in the resources section.
https://www.adacore.com/events

Dates	Events	Location/ Format
Nov 3, 2026	AdaCore Tech Day	Paris, in person
Jan 26-28, 2027	Embedded Systems Summit	Silicon Valley, in person
Jan 30-31, 2027	FOSDEM	Brussels, in person
Mar 16-18, 2027	Embedded World	Nuremberg, in person

Ada SPARK Office Hours

The Ada SPARK Office Hours are a place to chat with Ada and SPARK enthusiasts, pose your questions and learn from others. More information and the link to join the virtual meeting is on the Ada SPARK Office Hours page on our website.

The next office hours are:

- 10am-11am EDT | Friday, November 6, 2026
- 10am-11am EDT | Friday, November 20, 2026

Training

We offer **Public AdaCore Training** throughout the year, making our classroom experience with expert instructors accessible for teams of all sizes. This course is a ***cost-effective*** way to access our most popular language courses: **Ada Essentials** and **Rust Essentials**.

Our remaining dates for 2026 are:

- **Rust**
  - **US:** November 16 - 18

Our dates for 2027 are:

- **Ada**
  - **US:** March 22 - 26, September 20 - 24
  - **EU:** January 18 - 22, June 7 - 11
- **Rust**
  - **US:** January 19 - 21, June 8 - 10
  - **EU:** March 23 - 25, September 21 - 23

We look forward to seeing you there. See more information here.

If you'd like to take your training a step further, schedule a private session, or have a more critical time-based need, we offer Enterprise Training solutions for Ada, Rust, and our most popular tool suites (CodeSonar for Ada and GNAT DAS). You can explore all of our options here.

If you have questions about any of the technologies or services mentioned above, please reach out to your Account Manager or email us at info@adacore.com

