AdaCore

DIGEST

JULY 2026

AdaCore

DIGEST

Newsletter 07/07

In This Issue

The theme of this issue of the AdaCore Digest is the use of artificial intelligence in developing high-integrity software. We cover topics such as how to get started with LLM skills, how to use LLMs to convert C code to SPARK code at mode Silver, and much more.

There is, of course, an update for our release 26.2 in this issue as well!

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

26.2 Release

The 26.2 release is the last of the 26.x series and includes fixes but no new features for most AdaCore solutions. However, there are a couple of exceptions:

CodeSonar 9.2

CodeSonar 9.2, released alongside the 26.2 release, includes a long list of new features across the board. The top ones worth mentioning are:

- The AI-enabled warning explanation and remediation. This release features an MCP server that allows users to drive CodeSonar from their own LLM instance. Static analysis and the remediation of the warnings it flags are now part of the conversation with your LLM agent. You can ask the LLM to analyze your source code and provide a list of findings, and then ask the agent to iterate over the findings till the warnings are resolved. A short video is available from the AdaCore website.
- To improve scalability, CodeSonar now supports distributed parsing via Bazel. If your projects use Bazel as their build infrastructure, CodeSonar can be integrated into that infrastructure, with support for distributed builds, build avoidance, and caching of compile results.
- Improved coverage of C++ coding standards, including MISRA C++ and AUTOSAR. CodeSonar continues to follow the key coding standards in the high-integrity software field.
- We have been making great progress on integrating GNAT Static Analysis Suite into CodeSonar. It is not completely release-ready yet; it is scheduled for the 27.0 release. If you want a sneak peek, then you can reach out to your AdaCore sales engineer, and they can provide you with a demonstration.

- Significant improvement in the coverage of Kotlin, a language that is becoming increasingly important in the automotive in-vehicle-infotainment field.

These, of course, along with the minor bug fixes, upgrades of dependent open-source packages, runtimes for Java and C#, and the like, always go into a CodeSonar release.

Visual Studio Code Enhancements

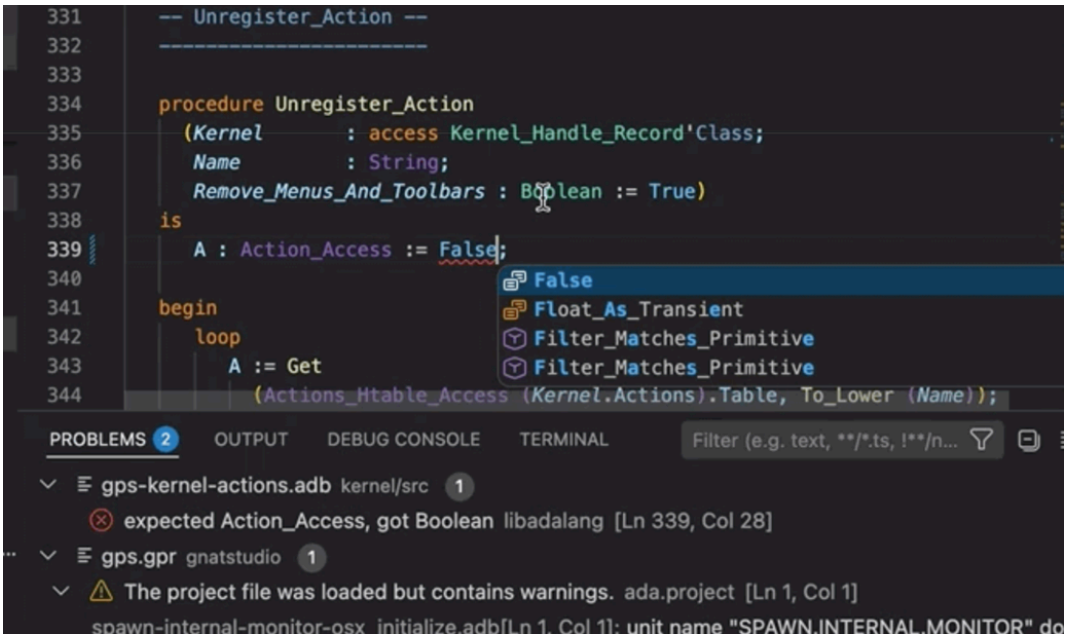
Enhancements to Visual Studio Code are delivered independently of the AdaCore GNAT Pro toolchain releases. Features are shipped more frequently when ready and can be picked up via the VS Code Marketplace ecosystem.

Some of the new features that you may wish to try out are the following:

GNATformat has new formatting options in the VS Code integration that allow the user to override the layout using a JSON file, for example, to specify how to format particular nodes, such as Aspects.

The **Project View Panel** in the VS Code explorer is now available and is driven by the GNAT Project (GPR) file and it lists the project as a tree, grouping source files by their project and source directories.

Dynamic highlighting of syntax errors provides developers with rapid, LibAdaLang-driven feedback on incorrect variable type assignments, use of undeclared names, invalid operators, and other errors in VS Code.



The goal of these features is to make VS Code an even more powerful software development environment for Ada and SPARK-based projects. Let us know what you think of these features. The development team is always looking for direct user feedback.

GNATpolyglot: How to Develop Multi-Language Software Development

We mentioned GNATpolyglot in our previous AdaCore Digest, and the engineering team at AdaCore has been making great strides in its implementation. As a refresher: GNATpolyglot accelerates multi-language software development by automating the manual, time-consuming process of generating and maintaining bindings that connect C++, Java, and Rust code to Ada code.

GNATpolyglot generates a language-agnostic representation of a piece of Ada code, the proxy, and from that proxy, bindings can be generated for all the supported languages. This proxy allows users to call into Ada programs from multiple other languages.

GNATpolyglot now allows software developers to combine not just C++ and Ada, but also to call Ada code from a Rust or Java application, and more languages can be added. GNATpolyglot is moving from private beta and to now being available for all GNAT Pro customers as a public beta.

GNATpolyglot for Java is not quite at the feature parity with AJIS (Ada-Java Interfacing Suite) yet; it supports numeric types, enum types, array types, record types, cross-language inheritance subprogram calls (with in and out parameters), global variables, and exception handling, including cross-language propagation.

The benefit of the Java support in GNATpolyglot over AJIS is that it uses modern Java, and the proxy architecture makes it easier to update and add additional languages as customer needs evolve.

On the C++ front, Rohan Kingan, one of our Sales Engineers, recently built a neat visualization showing that his code using Dear ImGui (https://github.com/ocornut/imgui) was calling into a piece of decision logic in Ada.

An ECU (Electronic Control Unit) related Ada package can easily be turned into a quick GUI using GNATpolyglot and the C++ proxy.

Reach out to your AdaCore sales contact if you are interested in early access to GNATpolyglot to try it in your environment.

AdaCore and AI

High-integrity software runs the world quietly; traffic lights, point-of-sale terminals, cars, trains, airplanes. We rarely notice it until it fails. And when it fails, the consequences are real: breaches of privacy, financial loss, injury, and even loss of life.

That's why people refer to software for these systems as high-integrity software, and high-integrity software is built differently. It is rigorously reviewed and thoroughly tested before it is deployed. Building high-integrity software takes more focus, more time, more people, and hence costs more. Building to a standard of high integrity keeps us safer, but the increased cost creates tension: we want more high-integrity, safe, and secure software, yet too often the economics don't allow it to be built or bought. The result is a world less safe than it could be. AdaCore believes that AI has the potential to reverse these economics. Using AI, we should be able to deliver safer, more secure software faster and at lower cost.

In the production of such high-integrity software, AI serves as a powerful tool, enabling human engineers to work more efficiently and effectively. Rather than replacing human engineers, AI can automate many of the steps required to produce and certify high-integrity software. Critically, however, AI does not replace human judgment, which is essential throughout the engineering process. Nor does AI replace human accountability for the software produced, which will be demanded by the humans exposed to the operation of the software, and is therefore an essential acceptance criterion for the production of the software.

Internally at AdaCore, we're applying AI thoughtfully, with clear, enforced processes. We will be on the lookout for opportunities to pass on lessons we've learned to you through our AI offerings.

For our customers, AdaCore's tools are the backbone of their high-integrity software development processes. We are committed to continuing to support all our tools. Whether you are integrating AI deeply into your workflows or continuing your established practices, our focus remains the same: delivering the best software development tools, whatever approach you take. For customers who are embracing agentic AI, we provide the missing piece that lets you take agent-produced artifacts safely into production: deterministic trust, end-to-end.

AdaCore uniquely provides a one-stop shop for your high-integrity software development. Our tools provide you with formal methods, MC/DC coverage, strong static analysis, smart fuzzing, testing, and broad targeting of relevant high-integrity embedded platforms. By using our tools, you restore confidence in software coauthored by agents.

We are working to refine our offering to maximize the benefits you'll see when you use our tools together with AI. As we work, we are releasing the skills we author as open source on GitHub at https://github.com/AdaCore/skills. This fall, we'll release a demonstration that illustrates the role deterministic trust can play in agentic workflows.

The remainder of this section of the AdaCore Digest provides a brief overview of LLM-related topics the team has been working on.

Getting Started with Ada SPARK Through LLM Skills

Agentic AI-based systems are really good at solving problems. Say, for example, you ask it to write a piece of software, in any language, really. It has the `knowledge` as to how to generate code, it will figure out how to get a compiler, then how to compile that code (assuming it needs a compiler), and finally, to solve compiler errors.

That is all nice and good. However, all that trial and error takes time, costs tokens, and is unpredictable. The LLM is re-inventing the wheel every time. An LLM SKILL is guidance that you can provide to an LLM on how to do specific things. It is the closest thing to 'downloading' a capability into a `brain`.

An LLM skill starts with a high-level description of a goal that the SKILL can help an agent with. For example, *'This skill should be used when the user wants to "use Alire", "install Alire", "create an Alire project", "add a crate", "set up tests for an Alire crate", "add AUnit / GNATtest to an Alire project", or otherwise refine the crate structure of an Ada project'*.

That is verbatim from the Alire SKILL, the SKILL that helps an LLM install and use Alire. SKILLs use progressive disclosure. That means the index page of the skill includes the goal and pointers to more information, for example, how to *'build'* with Alire.

AdaCore has published several skills at https://github.com/adaCore/skills. That location is a plugin directory you can add to the marketplace in your LLM, assuming your LLM supports such a concept, and it will be automatically kept up to date.

In that location, you will find the **Alire** skill, which gets you started with an Ada project, including installing Alire, and it can even create an embedded project. You can ask your LLM to *`create a Hello World project on the Raspberry Pi Pico 2W and lift it to SPARK Mode Silver,`* and the LLM will install Alire, the cross compilers, the runtime, GNATprove, and other tools and crates that it needs.

You will also find the **GNATprove** skill; this skill is instrumental if you want to write good SPARK code, or if you want to lift existing Ada code to SPARK. It contains a number of heuristics, best practices, and idioms that will help the LLM create idiomatic, elegant SPARK code. This skill was used extensively by Tony Aiello in his blog on Guidance, Navigation, and Control of the SPARK Mars Rover.

Then there is the **GNATfuzz skill**, for when you are looking to fuzz an application to find dangerous paths through the program. It helps the LLM understand the limitations of fuzzing without having to dig through GNATfuzz's entire documentation.

Lastly, there is the **GNATtest** skill for generating test cases for your Ada application. Like the GNATfuzz skill, this helps guide the LLM to quickly generate test cases and define the structure of your test setup.

More skills may be added in the near future, and we are, of course, always looking for suggestions on how to improve these skills. Star the repo if you are using the skills, and you think they are useful for your day-to-day use cases.

Translating C to Ada

Few software projects start from scratch. Most software projects start from an existing piece of software that needs to be augmented, modified, or extended. When that software is built in C, you are starting your project with technical debt from day one. Even if the software has been in heavy use for years, even if it's certified, as you start modifying it, new paths through the code can trigger latent bugs that have been in the code base for years, or memory corruption that was just never noticed.

Translation of that code from C to idiomatic Ada has been of interest to developers for years. It would strengthen the foundation, providing a secure base for the new project.

However, there was never an automated way to do this translation. It is a manual effort that requires expert personnel to perform the migration. This makes it slow and expensive, and hence often not an option for teams that have to move fast. The term idiomatic Ada is important in this translation. The code needs to be good Ada code, not C code, but then in Ada. The code needs types and subtypes and often requires a rethink of pointer-based data structures.

The frontier LLMs change the equation completely. The newest models can ingest C and produce fairly good, idiomatic Ada. To make this process successful, you need a measure of correctness, such as a good, comprehensive set of test cases.

To make it easier to convert entire applications, you can start by translating a single component. Convert from C to Ada, compile and link it back into the existing application. Re-run the test cases and make sure that nothing is broken. If it is, feed the input back into the LLM and ask it to fix it.

A bit more detail and a specific test case using the open-source curl software are available on the AdaCore blog site.

Lifting Ada to SPARK Silver or Gold

SPARK allows Ada code to be augmented with pre-conditions, post-conditions, and other contracts, and formal methods to be inserted straight into the program, in the language of the program. These contracts enable the mathematical proof of the program's properties. A property could be that the elements in an array are always sorted, or a property could be that there are no runtime errors, such as type overruns, buffer overruns/underruns, and the like. We call this latter example the Absence of Runtime Errors, or the Absence of Undefined Behavior.

To describe these different properties, we often use the terms Silver, Gold, and Platinum to describe increasing levels of correctness of an application. Platinum is the highest level, with the most properties proven. More information about the SPARK levels can be found in this paper authored by Thales and AdaCore.

Achieving SPARK Model Silver is a fantastic achievement for a piece of software. It means that you can statically prove that the application can never crash. Think about that for a second. Statically, before runtime. Think about how many hours your team has spent writing test cases to test all the paths through the source code. How often have they debugged an obscure error case in your application?

Gone. Forever.

Customers have reported up to 70% fewer defects in SPARK Silver code than in C & C++. That is a significant saving.

Achieving these savings by using SPARK Silver used to require software engineers who understand formal methods and your code. Achieving Silver used to be additional work, besides writing the application. Used to. Writing contracts is another area where LLMs excel.

If you have a piece of Ada code, you can now ask your favorite LLM to lift that code to SPARK mode Silver, and the LLM can perform that work for you. With the right heuristics, as described in the GNATprove skill mentioned in an earlier section.

Especially if you have elaborate test cases for your source code, then the GNATprove tool will be able to do a fantastic job lifting your code from Ada to SPARK without creating regressions in your code. There is a great article on the AdaCore website that walks through a sample use case for using AI to lift code to SPARK Silver.

The tools are open source; you can try this yourself instantly in your own code.

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:

- Proving asymptotic time complexity bounds with SPARK https://www.adacore.com/blog/proving-asymptotic-time-complexity-bounds-with-spark
- Formally verified hashed sets in Ada SPARK https://www.adacore.com/blog/formally-verified-hashed-sets-in-ada-spark
- Reasoning about linked structures in an array https://www.adacore.com/blog/reasoning-about-linked-structures-in-an-array
- Information hiding and context management in SPARK https://www.adacore.com/blog/information-hiding-and-context-management-in-spark

If you are interested in learning more about Rust:

- Adding libtest support to GNAT Pro for Rust https://www.adacore.com/blog/running-cargo-test-on-bare-metal-adding-libtest-support-to-gnat-pro-for-rust
- Getting started in functional safety with Rust https://www.adacore.com/blog/getting-started-in-functional-safety-with-rust

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
Sept 15, 2026	AdaCore Tech Day	Boston, in person
Sept 17, 2026	AdaCore Tech Day	Grand Rapids, in person
Sept 29-30, 2026	North American SPICE Conference	Michigan, in person
Oct 13, 2026	High Integrity Software Conference (HISC)	Birmingham, in person
Nov 3, 2026	AdaCore Tech Day	Paris, in person

Ada SPARK Office Hours
Office hours is a new initiative as part of our community outreach. They are held every 2 weeks on Friday from 10am-11am EDT. These are unstructured 'drop-in' calls where anyone doing anything with Ada and SPARK can call in and suggest topics to discuss. The first couple of sessions were very popular, and we discussed embedded hardware abstractions for bare board runtimes, AI, LLM workflows, and more. The list of upcoming sessions, complete with the Google Meet link, is available on the website mentioned above. Notes and recordings are posted on forum.ada-lang.io.

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:

- **Ada**
  - **US**: September 14 - 18
- **Rust**
  - **EU**: September 22 - 24
  - **US**: November 16 - 18

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 (GNAT SAS 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