GNAT Pro for Rust User's Guide

Release 27.0w

Apr 28, 2026

This page is intentionally left blank.

GNAT Pro for Rust, The AdaCore Rust Development Environment

Version 27.0w
Date: Apr 28, 2026

This page is intentionally left blank.

CONTENTS

CHAPTER

ONE

ABOUT THIS GUIDE

GNAT Pro for Rust is a toolchain for developing software in the Rust programming language. The product provides professional support for a specified version of the Rust toolset; it is not a fork of the Rust programming language or the Rust tools. For more information, see *Getting Started with GNAT Pro for Rust* (page 9).

This guide documents how to install the product and how to build, run, and debug both native and cross-compiled applications. It also shows how to build mixed-language programs (Rust and Ada), which is supported if you have subscriptions to both *GNAT Pro for Rust* and *GNAT Pro for Ada*.

GNAT Pro for Rust 27.0w implements version 1.93.0 of the Rust toolchain, for the 2021 edition of the Rust language, with additional fixes for the following CVEs:

- CVE-2024-43402

1.1 What This Guide Contains

This guide contains the following chapters:

- *Getting Started with GNAT Pro for Rust* (page 9) describes what the product includes, which platform configurations are supported, how to install the product, and how to verify that the installation was successful.
- *Developing Mixed-Language Projects* (page 15) describes how to build programs that contain code in both Rust and Ada.
- *Platform-Specific Information* (page 23) describes product details and program build/debug characteristics that are specific to the various native and embedded target configurations.

Appendix *Apache License* (page 31) contains the license for this document.

1.2 What You Should Know before Reading This Guide

This guide assumes a basic familiarity with the Rust language and toolset.

1.3 Related Information

For references to on-line resources for the Rust language and tools, please see *Documentation* (page 11).

1.4 Conventions

- Commands that are entered by the user are shown as preceded by a prompt string comprising the `$` character followed by a space.
- Regardless of whether the host platform is Linux or Windows, paths are shown with the `/` character as the directory separator; e.g., `my_package/src/main.rst`.

CHAPTER

TWO

GETTING STARTED WITH *GNAT PRO FOR RUST*

GNAT Pro for Rust is a complete development environment for the Rust programming language, supporting both native builds and cross compilation to embedded targets. The product is not a fork of the Rust programming language or the Rust tools. Instead, *GNAT Pro for Rust* is a professionally supported build of a selected version of `rustc` and other core Rust development tools that offers stability for professional and high-integrity Rust projects. Critical fixes to *GNAT Pro for Rust* are upstreamed to the Rust community, and critical fixes made by the community to upstream Rust tools are backported as needed to the *GNAT Pro for Rust* code base.

A product subscription includes items and licensing permissions that are specifically geared to professional Rust projects:

- *Front-line product support.* With AdaCore's web-based interface, customers receive accurate, helpful, and rapid responses to questions about any aspect of the product or the Rust language. The *GNAT Pro for Rust* developers themselves deliver the support, and the company's report tracking system ensures a full audit trail.
- *Predictable release schedule.* AdaCore offers product stability, issuing two releases per year, while also making updates available if needed to repair critical defects.
- *Support for mixed-language (Ada and Rust) development*. Both Ada and Rust are suitable languages for developing high-assurance software, providing memory safety and enforcing a variety of other checks to avoid vulnerabilities. Customers with a subscription to *GNAT Pro for Ada* can use *GNAT Pro for Rust* to write code that combines Rust and Ada.
- *Freely Licensed Open Source Software (FLOSS) licensing.* The product's flexible licensing permits customers to develop software covering a wide range of deployment scenarios and in particular allows distributing software binaries whose source code is proprietary and/or classified.

2.1 Supported Platforms

GNAT Pro for Rust is supported for native development on 64-bit Intel x86 Linux and Windows hosts, and for cross compilation to several 64-bit ARM target configurations:

- Windows host, AArch64 Bare Metal target
- Linux host, AArch64 Bare Metal target
- Windows host, AArch64 Embedded Linux target
- Linux host, AArch64 Embedded Linux target
- Linux host, AArch64 QNX 8.0 target
- Windows host, AArch64 VxWorks RTP and DKM target
- Linux host, AArch64 VxWorks RTP and DKM target

2.2 Product Components

2.2.1 Tools

GNAT Pro for Rust includes the Rust development toolsuite together with AdaCore's multi-language build utility:

- **`cargo`** - Rust package manager
- **`rustc`** - Rust compiler
- **`gdb`** - Rust-aware debugger
- **`rust-analyzer`** - Rust language server / IDE integration tool
- **`clippy`** - Rust linter
- **`rustdoc`** - Rust documentation generator
- **`rustfmt`** - Rust code formatter
- **`gprbuild`** - AdaCore's multi-language build tool

2.2.2 Libraries

For `no_std` environments, *GNAT Pro for Rust* supplies the following crates:

- alloc
- core

The following targets are `no_std` environments:

- Windows host, AArch64 Bare Metal target
- Linux host, AArch64 Bare Metal target
- Windows host, AArch64 VxWorks DKM target
- Linux host, AArch64 VxWorks DKM target

For `std` environments, the product supplies the Rust Standard Library, which comprises the above crates plus the following:

- proc_macro
- std
- test

The following targets are `std` environments:

- 64-bit Intel x86 Linux
- 64-bit Intel x86 Windows
- Windows host, AArch64 Embedded Linux target
- Linux host, AArch64 Embedded Linux target
- Linux host, AArch64 QNX 8.0 target
- Windows host, AArch64 VxWorks RTP target
- Linux host, AArch64 VxWorks RTP target

2.2.3 Examples

Source code examples are located in subdirectories of *INSTALL_DIR*`/share/examples/rust/` where *INSTALL_DIR* is the directory in which the installation was performed.

For products that support the Rust Standard Library, subsection *Running a Simple Rust Program* (page 12) provides instructions on how to build and run the default `Hello, world` program that comes with the Rust environment. Further examples for both `std` and `no_std` environments may be found in *Platform-Specific Information* (page 23).

2.2.4 Documentation

GNAT Pro for Rust includes copies of the `rust-lang.org` documentation of the Rust language and toolset:

- The Rust Reference
- The Rust Edition Guide
- The Rust Embedded Book
- The Rustonomicon
- The Rustc Book
- The Cargo Book
- The Clippy Book
- The Rustdoc Book

2.3 Installing the Product

On Windows, execute the provided installer and follow its instructions.

On Linux, unpack the downloaded archive, either using a GUI or at the command line by executing:

```
$ tar xf archive.tar.gz
```

Install *GNAT Pro for Rust* by executing:

```
$ ./doinstall [INSTALL_DIR]
```

where *INSTALL_DIR* is an optional installation directory of your choice. If an installation directory is not provided, the `doinstall` script will prompt you for one.

Export the appropriate environment variables as directed by the `doinstall` script.

2.3.1 Rust Analyzer VS Code Extension

Start VS Code.

Uninstall any previous `rust-analyzer` extensions.

Press Ctrl+Shift+P and select "Extensions: Install from VSIX…".

Navigate to *INSTALL_DIR*`/share/vscode` and select the `.vsix` file.

2.4 Running a Simple Rust Program

This section is specific to native Linux and native Windows. For instructions on building and running examples that execute on `no_std` targets or require cross-compilation, please see *Platform-Specific Information* (page 23).

We will start by creating a simple "Hello World" program.

2.4.1 Creating

In a directory of your choice, execute:

```
$ cargo new hello_world
```

cargo is the Rust package manager, the entry point of *GNAT Pro for Rust*. It offers an extensive set of commands and configurations. Consult Cargo Commands and Cargo Reference for further details.

By default the **cargo new** command will create a package for an executable program (binary crate) that displays the classical `Hello, world!` greeting.

The command should create a directory `hello_world`, with the following key files:

- File `Cargo.toml` is the manifest file used by **cargo**. The manifest format is quite powerful as it allows you to specify the identity of your crate, its dependencies, target-specific details, profiles, workspaces, and much more. Consult the Manifest Format for further details.
- File `src/main.rs` contains the source code of the program:

```
fn main(){
    println!("Hello, world!");
}
```

Navigate into directory `hello_world`.

2.4.2 Building

To build the program, execute:

```
$ cargo build
```

The command should terminate successfully with output similar to the following:

```
Compiling hello_world v0.1.0 (<path/to/program>)
Finished `dev` profile [unoptimized + debuginfo] target(s) in <time>
```

2.4.3 Running

To run the program, execute:

```
$ cargo run
```

The command should terminate successfully with output similar to the following:

```
Finished `dev` profile [unoptimized + debuginfo] target(s) in <time>
Running `target/debug/hello_world`
Hello, world!
```

2.4.4 Debugging

Start **gdb** by executing:

```
$ gdb target/debug/hello_world
```

Place a breakpoint on `main.rs` line 2, run the program, and step into the call to `println!` by executing:

```
(gdb) break main.rs:2
(gdb) run
(gdb) step
```

The debugger should display output similar to the following:

```
core::fmt::Arguments::new_const (pieces=&[&str](size=1) = {...})
```

Quit **gdb** by executing:

```
(gdb) quit
```

2.5 Running the Rust Tools

2.5.1 Rust Analyzer

Navigate to the Rust Analyzer example in *INSTALL_DIR*`/share/examples/rust/rust-analyzer` and follow the instructions in the `README.rst` file.

2.5.2 clippy

clippy is the Rust linter. It offers a multitude of checks and several levels of diagnostic severity. Consult Clippy Book for further details.

Navigate to the **clippy** example in *INSTALL_DIR*`/share/examples/rust/clippy` and follow the instructions in the `README.rst` file.

2.5.3 rustdoc

rustdoc is the Rust documentation generator. It produces documentation from doc comments found in Rust sources. Consult Rustdoc Book for further details.

Navigate to the **rustdoc** example in *INSTALL_DIR*`/share/examples/rust/rustdoc` and follow the instructions in the `README.rst` file.

2.5.4 rustfmt

rustfmt is the Rust code formatter. It changes the layout of Rust source code accoding to style guidelines. Consult https://rust-lang.github.io/rustfmt/?version=v1.7.0&search= for further details.

Navigate to the **rustfmt** example in *INSTALL_DIR*`/share/examples/rust/rustfmt` and follow the instructions in the `README.rst` file.

This page is intentionally left blank.

CHAPTER

THREE

DEVELOPING MIXED-LANGUAGE PROJECTS

With a *GNAT Pro for Ada* subscription, you can use the `gprbuild` tool in *GNAT Pro for Rust* to create applications that combine Ada and Rust. This chapter illustrates how to write a program with an Ada main procedure that invokes Rust functions from static and dynamic library crates. Combining Ada and Rust involves using a C interface on the Ada side and the `core::ffi` or `std::ffi` (Foreign Function Interface) module in the Rust code.

This section is specific to native Linux and native Windows. For instructions on building and running mixed language projects that execute on `no_std` targets or require cross-compilation, please see *Platform-Specific Information* (page 23).

3.1 Current Limitations

This version of *GNAT Pro for Rust* provides tooling support for only the "Ada main with Rust libraries" use case. Future versions should offer complete mixed language support, including support for a Rust main with Ada libraries. Regardless, we have explained how to manually develop such an application in section *Rust Main with Ada Library* (page 18).

3.2 Ada Main with Rust Library

We will start by creating a Rust library and an Ada main, which will output the greetings `Hello from Rust!` and `Hello from Ada!`, respectively.

The directory structure we are about to create is not mandatory for mixed language development. It simply illustrates the division of the source code across languages.

In a directory of your choice, create the following directory structure:

```
ada_with_rust/
+-- ada/
+-- rust/
```

3.2.1 Rust Library

Creating

Navigate to directory `ada_with_rust/rust` and execute:

```
$ cargo new --lib hello_from_rust
```

By default, the **`cargo new --lib`** command will create a package for an `rlib` (Rust static library) that adds two numbers.

The command should create directory `hello_from_rust`, with the following key files:

- `Cargo.toml` is the manifest file used by **cargo**.
- `src/lib.rs` contains the source code of the library.

However, an `rlib` is not suitable for mixed language development.

Navigate to directory `hello_from_rust`.

To convert the library into either a `cdylib` (native dynamic library) or a `staticlib` (native static library), edit file `Cargo.toml` as follows:

```
[package]
name = "hello_from_rust"
version = "0.1.0"
edition = "2024"

[lib]
crate-type = ["staticlib"]

[dependencies]
```

The `crate-type` property indicates the package kind. Consult the Manifest Format for further details.

Edit file `hello_from_rust/src/lib.rs` as follows:

```
#[unsafe(no_mangle)]
pub extern "C" fn hello_from_rust() {
    println!("Hello from Rust!");
}
```

Functions that are intended for use by other languages through the C ABI must be subject to attribute `#[unsafe(no_mangle)]` and marked as `extern "C"`. Attribute `#[unsafe(no_mangle)]` causes the related item to be publicly exported from the produced library or object file. ABI specification `extern "C"` indicates that the related item uses the C ABI. Consult Foreign Function Interface for further details.

Building

To build the Rust library, execute:

```
$ cargo build --release
```

The command should terminate successfully with output similar to the following:

```
Compiling hello_from_rust v0.1.0 (<path/to/program>)
Finished release [optimized] target(s) in <time>
```

Command line argument `--release` is mandatory for mixed-language development in this version of *GNAT Pro for Rust*.

3.2.2 Ada Main

Creating

Navigate to directory `ada_with_rust/ada`, and create file `hello_from_rust.gpr` with the following content:

```
library project Hello_From_Rust is

   for Languages use ("Rust");
   for Library_Name use "hello_from_rust";
```

(continues on next page)

(continued from previous page)

```
   for Library_Dir use Project'Project_Dir & "../rust/" & Project'Library_Name;
   for Source_Dirs use (Project'Library_Dir & "/src");
   for Externally_Built use "true";

   package Naming is
      for Body_Suffix ("Rust") use ".rs";
   end Naming;

end Hello_From_Rust;
```

Each Rust library must have exactly one corresponding GPR project in this version of *GNAT Pro for Rust*.

The Rust library project must be configured as follows:

- Attribute `Languages` must be present, with value `("Rust")`.
- Attribute `Library_Name` must be the same as the name of the Rust package.
- Attribute `Library_Dir` must indicate the path to the Rust package, aka the *package root*.
- Attribute `Source_Dirs` must indicate the source directory within the package root.
- Attribute `Externally_Built` must be present, with value `"true"`.

Create file `hello_from_ada.gpr` with the following content:

```
with "hello_from_rust.gpr";

project Hello_From_Ada is

   for Languages use ("Ada");
   for Object_Dir use "obj";
   for Exec_Dir use ".";

   for Main use ("hello_from_ada.adb");

   for Target use "<target>";

end Hello_From_Ada;
```

where `<target>` is `x86_64-linux` on native Linux and `x86_64-windows64` on native Windows.

The Ada main project must be configured as follows:

- The project must "with" each Rust library project.
- Attribute `Target` must indicate the compilation target.

Create file `hello_from_ada.adb` with the following content:

```
with Ada.Text_IO; use Ada.Text_IO;

procedure Hello_From_Ada is
   procedure Hello_From_Rust with Import, Convention => C;

begin
   Put_Line ("Hello from Ada!");
   Hello_From_Rust;

end Hello_From_Ada;
```

Building

To build the Ada main, execute:

```
$ gprbuild -P hello_from_ada.gpr
```

The command should terminate successfully with output similar to the following:

```
Setup
   [mkdir]        object directory for project Hello_From_Ada
Compile
   [Ada]          hello_from_ada.adb
Bind
   [gprbind]      hello_from_ada.bexch
   [Ada]          hello_from_ada.ali
Link
   [link]         hello_from_ada.adb
```

Running

To run the Ada main, execute:

```
$ ./hello_from_ada
```

The command should terminate successfully with output similar to the following:

```
Hello from Ada!
Hello from Rust!
```

3.3 Rust Main with Ada Library

We will start by creating an Ada library and a Rust main, which will output the greetings `Hello from Ada!` and `Hello from Rust!`, respectively.

The directory structure we are about to create is not mandatory for mixed language development. It simply illustrates the division of the source code across languages.

In a directory of your choice, create the following directory structure:

```
rust_with_ada/
+-- ada/
+-- rust/
```

3.3.1 Ada Library

Creating

Navigate to directory `rust_with_ada/ada`, and create file `hello_from_ada.gpr` with the following content:

```
library project Hello_From_Ada is

   for Languages use ("Ada");
   for Library_Name use "hello_from_ada";
   for Library_Standalone use "encapsulated";
   for Library_Interface use ("Ada_Lib");
```

(continues on next page)

(continued from previous page)

```
   for Library_Dir use "lib";
   for Object_Dir use "obj";
   for Source_Dirs use (".");

end Hello_From_Ada;
```

The Ada library project must be configured as follows:

- Attribute `Languages` must be present, with value `("Ada")`.
- Attribute `Library_Name` must be present, where the value will be used in Rust main's `.cargo/config.toml` file.
- Attribute `Library_Standalone` must be present, with value `"encapsulated"`.
- Attribute `Library_Interface` must be present, where the value enumerates all units whose subprograms will be used by the Rust main.
- Attributes `Library_Dir`, `Object_Dir`, and `Source_Dirs` must be present, and indicate distinct subdirectories. The value of attribute `Library_Dir` will be used in Rust main's `.cargo/config.toml` file.

Create file `ada_lib.ads` with the following content:

```
package Ada_Lib is
   procedure Hello_From_Ada
     with Export, Convention => C, Link_Name => "hello_from_ada";
end Ada_Lib;
```

Subprograms intended for use by the Rust main must be declared as follows:

- Aspect `Export` must be present.
- Aspect `Convention` must be present, with value `C`.
- Aspect `Link_Name` must be present, where the value denotes the name of the corresponding importing function in the Rust main.

Create file `ada_lib.adb` with the following content:

```
with Ada.Text_IO; use Ada.Text_IO;

package body Ada_Lib is
   procedure Hello_From_Ada is
   begin
      Put_Line ("Hello from Ada!");
   end Hello_From_Ada;
end Ada_Lib;
```

Building

To build the Ada library, execute:

```
$ gprbuild -P hello_from_ada.gpr
```

The command should terminate successfully with output similar to the following:

```
Setup
   [mkdir]        object directory for project Hello_From_Ada
   [mkdir]        library directory for project Hello_From_Ada
```

(continues on next page)

(continued from previous page)

```
Compile
   [Ada]          ada_lib.adb
Build Libraries
   [gprlib]       hello_from_ada.lexch
   [bind SAL]     hello_from_ada
   [Ada]          b__hello_from_ada.adb
   [objcopy]      p__hello_from_ada_0.o
   [archive]      libhello_from_ada.a
   [index]        libhello_from_ada.a
```

3.3.2 Rust Main

Creating

Navigate to directory `rust_with_ada/rust` and execute:

```
$ cargo new --bin hello_from_rust
```

The command should create directory `hello_from_rust`, with the following key files:

- `Cargo.toml` is the manifest file used by **`cargo`**.
- `src/main.rs` contains the source code of the executable.

Navigate to directory `hello_from_rust`, and edit file `src/main.rs` as follows:

```
unsafe extern "C" {
    fn hello_from_ada();
}

fn main() {
    unsafe { hello_from_ada(); }
    println!("Hello from Rust!");
}
```

Subprograms exported by the Ada library must be imported as follows:

- An unsafe block with ABI `C` must be present.
- Each exported subprogram by the Ada library
  - Must reside within the unsafe block.
  - Must have a matching function declaration where the name of the function maches the value of aspect `Link_Name`.
  - Must be called within an unsafe block.

Create file `.cargo/config.toml` with the following content:

```
[target.<target>]
rustflags = [
   "-L../../ada/lib",
   "-lhello_from_ada"
]
```

where `<target>` is `x86_64-unknown-linux-gnu` on native Linux and `x86_64-pc-windows-gnu` on native Windows.

In order to automatically rebuild the Rust main whenever the Ada library changes, create build script `build.rs` with the following content:

```
fn main() {
    println!("cargo::rerun-if-changed=../../ada/lib/libhello_from_ada.a");
}
```

Building

To build the Rust main, execute:

```
cargo build
```

Running

To run the Rust main, execute

```
cargo run
```

The command should terminate successfully with output similar to the following:

```
Hello from Ada!
Hello from Rust!
```

This page is intentionally left blank.

CHAPTER

FOUR

PLATFORM-SPECIFIC INFORMATION

This chapter presents the platform-dependent instructions for using *GNAT Pro for Rust* to build, run, and debug Rust programs. It assumes that *INSTALL_DIR* is the directory in which the product has been installed.

4.1 Native Linux and Native Windows

4.1.1 Introduction

Native Linux and native Windows are targets which come with the Rust Standard Library consisting of:

- alloc
- core
- proc_macro
- std
- test

The targets are identified by the following triples:

AdaCore triple	Rust triple
x86_64-linux	x86_64-unknown-linux-gnu
x86_64-windows64	x86_64-pc-windows-gnu

4.1.2 Limitations

The *GNAT Pro for Rust* products for native Linux and native Windows have no known limitations.

4.1.3 Using *GNAT Pro for Rust*

sudoku

The sudoku example demonstrates development with the Rust Standard Library, in particular the organization of source code into modules, the implementation of comparison and formatting traits, the use of simple data structures, and more. Navigate to *INSTALL_DIR*/share/examples/rust/sudoku and follow the instructions in the README.rst file.

mouse-in-maze

The `mouse-in-maze` example demonstrates mixed language development with *GNAT Pro for Rust* and *GNAT Pro for Ada*. Navigate to *INSTALL_DIR*`/share/examples/rust/mouse-in-maze` and follow the instructions in the `README.rst` file.

4.2 AArch64 Bare Metal

4.2.1 Introduction

AArch64 Bare Metal is a `no_std` target which comes with the following Rust libraries:

- alloc
- core

AArch64 Bare Metal includes experimental versions of the following Rust libraries:

- std

This allows regular Rust development (i.e., without `no_std`) as far as applicable to the target device, backed by a bare-metal implementation of libc that is included in the product. Note that `std` functions which cannot work due to bare-metal constraints may panic at run time; for example, the absence of a file system implies that `std::file` is not usable.

AArch64 Bare Metal includes a crate that provides support for the AMD Zynq UltraScale+ MPSoC:

- zynqmp

The crate supports the use of `std` as described above.

The target is identified by the following triples:

AdaCore triple	Rust triple
`aarch64-elf`	`aarch64-unknown-none`

4.2.2 Limitations

GNAT Pro for Rust for AArch64 Bare Metal has no known limitations.

4.2.3 Using *GNAT Pro for Rust*

aarch64-elf

The `aarch64-elf` example demonstrates `no_std` development, in particular the use of FFI (Foreign Function Interface) to interface with C code, the definition of a global allocator in order to use heap-allocated data structures, and the handling of panics. Navigate to *INSTALL_DIR*`/share/examples/rust/aarch64-elf` and follow the instructions in the `README.rst` file.

sudoku

The `sudoku` example demonstrates development with the experimental version of the Rust Standard Library, in particular the organization of source code into modules, the implementation of comparison and formatting traits, the use of simple data structures, and more. Navigate to *INSTALL_DIR*`/share/examples/rust/sudoku` and follow the instructions in the `README.rst` file.

arithmetic

The `arithmetic` example demonstrates mixed language development with *GNAT Pro for Rust* and *GNAT Pro for Ada*. Navigate to *INSTALL_DIR*`/share/examples/rust/arithmetic` and follow the instructions in the `README.rst` file.

4.2.4 Using the `zynqmp` crate

The `zynqmp` crate is located in *INSTALL_DIR*`/share/cargo/registry`. You can use the crate by adding `zynqmp` as a dependency to your project:

```
[dependencies]
zynqmp = { path = "{INSTALL_DIR}/share/cargo/registry/zynqmp-{VERSION}" }
```

Further information on how to use this crate can be found in the zynqmp documentation.

4.3 AArch64 Linux

4.3.1 Introduction

AArch64 Linux is a target which comes with the Rust Standard Library consisting of:

- alloc
- core
- proc_macro
- std
- test

The target is identified by the following triples:

AdaCore triple	Rust triple
`aarch64-linux`	`aarch64-unknown-linux-gnu`

4.3.2 Limitations

GNAT Pro for Rust for AArch64 Linux has no known limitations.

4.3.3 Using *GNAT Pro for Rust*

sudoku

The `sudoku` example demonstrates development with the Rust Standard Library, in particular the organization of source code into modules, the implementation of comparison and formatting traits, the use of simple data structures, and more. Navigate to *`INSTALL_DIR`*`/share/examples/rust/sudoku` and follow the instructions in the `README.rst` file.

mouse-in-maze

The `mouse-in-maze` example demonstrates mixed language development with *GNAT Pro for Rust* and *GNAT Pro for Ada*. Navigate to *`INSTALL_DIR`*`/share/examples/rust/mouse-in-maze` and follow the instructions in the `README.rst` file.

4.4 AArch64 QNX 8.0

4.4.1 Introduction

AArch64 QNX 8.0 is a target which comes with the Rust Standard Library consisting of:

- alloc
- core
- proc_macro
- std
- test

The target is identified by the following triples:

AdaCore triple	Rust triple
`aarch64-qnx`	`aarch64-unknown-nto-qnx800`

4.4.2 Limitations

GNAT Pro for Rust for AArch64 QNX 8.0 has the following limitations:

IPv6 Networking

The support of IPv6 networking cannot be guaranteed.

4.4.3 Using *GNAT Pro for Rust*

sudoku

The `sudoku` example demonstrates development with the Rust Standard Library, in particular the organization of source code into modules, the implementation of comparison and formatting traits, the use of simple data structures, and more. Navigate to *INSTALL_DIR*`/share/examples/rust/sudoku` and follow the instructions in the `README.rst` file.

mouse-in-maze

The `mouse-in-maze` example demonstrates mixed language development with *GNAT Pro for Rust* and *GNAT Pro for Ada*. Navigate to *INSTALL_DIR*`/share/examples/rust/mouse-in-maze` and follow the instructions in the `README.rst` file.

4.5 AArch64 VxWorks DKM

AArch64 VxWorks DKM is a `no_std` target which comes with the following Rust libraries:

- alloc
- core

The target is identified by the following triples:

AdaCore triple	Rust triple
`aarch64-vx7r2`	`aarch64-wrs-vxworks-dkm`

4.5.1 Limitations

GNAT Pro for Rust for AArch64 VxWorks DKM has the following limitations:

No executables

The generation of executables is disabled. It is recommended to create static libraries.

Dynamic Libraries

Dynamic libraries are not supported.

4.5.2 Using *GNAT Pro for Rust*

Prerequisites

Make sure you have a VxWorks Source Build available, and that the `VSB_DIR` environment variable points to the directory that contains it.

Debugging

Debugging support is provided via `vxgdb`.

For detailed instructions, please refer to the official Wind River documentation:

- General instructions: Remote Debugging with gdbserver
- DKM-specific instructions: Debugging Kernel Tasks from a DKM

vxworks-dkm

The `vxworks-dkm` example demonstrates development of VxWorks Downloadable Kernel Modules. In particular, it shows which symbols need to be defined to create a minimal working module and which commands are needed to build it, and how to set up a basic panic handler in a `no_std` context. Navigate to *INSTALL_DIR*`/share/examples/rust/vxworks-dkm` and follow the instructions in the `README.rst` file.

4.6 AArch64 VxWorks RTP

4.6.1 Introduction

AArch64 VxWorks RTP is a target which comes with the Rust Standard Library consisting of:

- alloc
- core
- proc_macro
- std
- test

The target is identified by the following triples:

AdaCore triple	Rust triple
`aarch64-vx7r2`	`aarch64-wrs-vxworks-rtp`

4.6.2 Limitations

GNAT Pro for Rust for AArch64 VxWorks RTP has the following limitations:

Stack Overflow Handling

No stack overflow handler is available. A stack overflow will crash the RTP.

Backtrace Symbolization

No symbols will be shown in backtraces.

Dynamic Libraries

Dynamic libraries are not supported.

4.6.3 Using *GNAT Pro for Rust*

Prerequisites

Make sure you have a VxWorks Source Build available, and that the `VSB_DIR` environment variable points to the directory that contains it.

The following components are required by the Rust standard library:

- INCLUDE_TLS (included by default since VxWorks 7)
- INCLUDE_POSIX_PTHREAD_SCHEDULER (required for spawning threads)
- INCLUDE_POSIX_PIPES (required for inter-process communication)
- INCLUDE_SC_REALPATH (required for path canonicalization)

Debugging

Debugging support is provided via `vxgdb`.

For detailed instructions, please refer to the official Wind River documentation:

- General instructions: Remote Debugging with gdbserver
- RTP-specific instructions: Debugging RTP Tasks

sudoku

The `sudoku` example demonstrates development with the Rust Standard Library, in particular the organization of source code into modules, the implementation of comparison and formatting traits, the use of simple data structures, and more. Navigate to *`INSTALL_DIR`*`/share/examples/rust/sudoku` and follow the instructions in the `README.rst` file.

mouse-in-maze

The `mouse-in-maze` example demonstrates mixed language development with *GNAT Pro for Rust* and *GNAT Pro for Ada*. Navigate to *`INSTALL_DIR`*`/share/examples/rust/mouse-in-maze` and follow the instructions in the `README.rst` file.

This page is intentionally left blank.

APPENDIX

A

APACHE LICENSE

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. *Grant of Copyright License.* Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. *Grant of Patent License.* Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. *Redistribution.* You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. *Submission of Contributions.* Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. *Trademarks.* This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. *Disclaimer of Warranty.* Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. *Limitation of Liability.* In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to

use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. *Accepting Warranty or Additional Liability.* While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

This page is intentionally left blank.

INDEX