The Babel Programming Language

WehrWolff

February 4, 2026

Contents

Contents ii

I Manual 1

1 Babel 0.0 Documentation 2

1.1 Important Links . . . 2

1.2 Introduction . . . 2

1.3 Key Features of Babel . . . 3

2 Getting started 5

3 Constants 6

Part I

Manual

Chapter 1

Babel 0.0 Documentation

Welcome to the documentation for Babel 0.0.

Work in progress!

This documentation is for an unreleased, in-development, version of Babel.

Note

The documentation is also available in PDF format: TheBabelProgrammingLanguage.pdf.

1.1 Important Links

Below is a non-exhaustive list of links that will be useful as you learn and use the Babel programming language.

- Babel Homepage
- Download Babel
- Discussion forum
- Babel YouTube
- Find Babel Packages
- Learning Resources

1.2 Introduction

In the world of programming, there has always been a tension between control and abstraction. Developers have long sought languages that balance performance and flexibility, allowing them to efficiently solve problems while maintaining control over system resources. As software systems grow more complex, the demand for both low-level control and high-level ease of use has become even more crucial.

Babel is a general-purpose programming language designed to meet this challenge. It combines low-level control—such as direct memory management and system resource access—with the expressiveness and safety features of higher-level languages. This empowers developers to write efficient, performant, and maintainable code. Babel provides the raw performance needed for systems programming and other performance-critical tasks, while also remaining approachable for developers looking to build web applications, mobile apps, or general-purpose tools.

Babel is designed for both experienced developers who require fine-grained control over system resources and newcomers who need a smooth entry into the world of programming. It allows users to learn critical concepts such as pointers and memory management in a safe, accessible environment, making it ideal as a first language without sacrificing the power and flexibility expected from more advanced languages like C.

1.3 Key Features of Babel

1. **Multi-Paradigm Design**

   Babel supports a wide range of programming paradigms, providing developers with the flexibility to choose the most suitable approach for their projects. It combines the best features of various programming styles to create a powerful and adaptable environment:

   - **Object-Oriented Programming (OOP):** With class-based object-oriented features, Babel allows developers to structure their code in a modular and reusable way, promoting clean, maintainable designs.
   - **Procedural (Imperative) Programming:** For those who prefer a more straightforward approach, Babel supports traditional procedural programming, where the logic is organized into functions and executed in sequence.
   - **Functional Programming:** Babel supports lambda functions and higher-order functions, enabling functional programming techniques that allow for concise, expressive code and better abstraction.

   This multi-paradigm support allows Babel to adapt to different styles and project requirements, giving developers the freedom to choose the most effective approach to solve their problems.

2. **Various Compilation Models**

   Babel provides a flexible compilation model that enables both high performance and rapid iteration, accommodating different use cases and development styles:

   - **Ahead-of-Time (AOT) Compilation:** By default, Babel uses AOT compilation to generate optimized machine code before execution. This ensures that code is highly efficient, with minimal runtime overhead, making it ideal for performance-critical applications such as systems programming and real-time systems.
   - **Just-in-Time (JIT) Compilation:** In addition to AOT, Babel also supports JIT compilation, which allows the compiler to generate machine code at runtime. This can lead to more dynamic execution and is particularly useful for scenarios where runtime performance tuning is essential.
   - **Bytecode Generation:** Babel is designed to be compiled to bytecode in the future, offering the possibility of platform-independent code execution, similar to Java. This enables cross-platform compatibility and makes it easier to target different architectures.

3. **Seamless Interoperability**

   One of Babel's standout features is its ability to seamlessly integrate with existing codebases and libraries, ensuring broad compatibility across languages and platforms:

   - **C Integration:** Babel provides straightforward integration with C, allowing developers to call C functions directly from Babel code. This ensures that you can leverage existing C libraries and tools, making it ideal for system-level programming or projects that require specialized C libraries.
   - **LLVM IR Interoperability:** Babel is designed to work with any language that emits LLVM Intermediate Representation (IR), such as Rust, Swift, and more. This compatibility opens the door for developers to use libraries or code written in other languages that target the LLVM ecosystem.

- **Cross-Platform Compatibility:** Babel can be compiled for different platforms, from embedded systems to high-performance servers. Its ability to work across diverse environments makes it highly versatile and adaptable for a wide range of use cases.

Chapter 2

Getting started

Traditionally, the first program in any new programming language is one that displays the words “Hello world” on the screen. In Babel, this can be done using the `print` function. Here’s how you’d write it:

```
print("Hello world")
# Prints "Hello world" without a newline
```

This syntax should feel familiar if you've worked with other languages. In Babel, this line of code is a complete program — no need for complex setups or boilerplate. There's no `main()` task required to start, but if you prefer to explicitly define one, you can do so to mark the entry point of your program for larger projects.

```
task main(argc: Int, argv: vList<String>) => void!
    if argc > 1 then
        println() <| fmt() <| "Greetings {argv.join(', ')}!"
    else
        println() <| "Greetings Universe!"
    end
end
```

Whoa, there's a lot going on here now! This is just to show experienced programmers what is possible in Babel. We’ll cover all these concepts — from tasks, argument handling, the pipe operator, and string formatting — in upcoming sections. Don't worry if this seems like a lot to take in right now!

Chapter 3

Constants

They are cool.