

AFR-o

The Amlantis System Overview

Version 0.11

Markéta Lisová

July 27, 2020

Special Thanks To

Sabina Eva Lisová

Marie Strnadová

Ondřej Profant

Vojtěch Biberle

Contents

I	Introduction	1
1	A Brief Introduction to the Amlantis System	3
1.1	The Amlantis Languages	4
II	The Amlantis System Environment	5
2	Language Readers	7
2.1	Root Reader Abilities	8
III	The Amlantis System Tools	9
3	The Top-Level System or REPL – aml	11
4	The Compiler – amlc	13
5	The Runtime System – amlrun	15
6	The Debugger – amldebug	17
7	The Profiler – amlprof	19
8	The Builder – amlbuild	21
9	The Book Workspace – amlbook	23

Part I

Introduction

Chapter 1

A Brief Introduction to the Amlantis System

Amlantis System is a collection of specifications of programming and data definition languages and their related tools, runtimes and libraries.

Those specifications do not always require particular implementations, but attempt to avoid undefined behaviours as much as possible.

Amlantis also provides an open source reference implementation of those specifications, but it is indeed possible for other people to write their own implementations or forks¹ of this default implementation, probably focusing on optimising other aspects of the system, maybe exploring new options of future development to be pull-requested into the default implementation.

A Few Notes on the Name

Amlantis' name has quite some history. The project started being named *Coral*, but that collided with another language of a similar name, *CORAL 66*. Then it got renamed to *Gear*, but that again collided with another language of the same name, which seemed inactive at the time, *zippers/gear*. Then an idea was born and Aml was named *Amlantis*, which is whatever you want it to be. It could be a misspelling of *Atlantis*², it could be an acronym like *A ML Language*, or maybe even something like *A ML Language And Neat Technology Improvement System*, or maybe *Caml* without the *C*. For the meaning of the cryptic *ML* part, search for the *Standard ML* or *OCaml*.

¹Forks and pull requests are preferred way of help!

²Intentionally – because otherwise, it would be named Atlantis, but there is already a city of that name.

1.1 The Amlantis Languages

The Amlantis System is a home to more than only one programming language – in fact, it can be home to any number of languages, ones that Amlantis users can either extend from existing languages create or even create new ones entirely. Every language build within the Amlantis System shares a common type system and value models, and thus are interoperable.

The minimal Amlantis System contains these languages:

- Aml/Base, a minimal Lisp-like³ language that is easy to parse and limited in functionality.
- Aml/Core, an ML-like⁴ language that is easier to read and harder to parse, fully featured in functionality.
- Aml, an ML-like⁵ language that extends Aml/Core, is easier to read and the hardest one to parse, also fully featured in functionality, and introducing more grammar than its parent Aml/Core.

More languages may be added to this list over time.

³One might say that Aml/Base is a grandchild of Racket and Clojure.

⁴One might say that Aml/Core is a grandchild of OCaml, SML and F#.

⁵One might say that Aml is a grandchild of Aml/Core, Haskell and Ada.

Part II

The Amlantis System Environment

Chapter 2

Language Readers

Amlantis System provides multiple interfaces to talk to it with. Those interfaces would be:

- A REPL console, where user's text input is immediately read, evaluated, printed and requested again.
- Source code, where user's text input is stored for reuse in general, usually to either do some scripting, or building modules and complex applications.
- Bytecode or native (machine) code files, which originate from textual source code.

More ways to interact with the Amlantis System might be possible in the future.¹

Either way, for now, text-based interface to all Amlantis System tools require it to have a component that is able to read the text. We call them *Language Readers* (or shortly just *Readers* for the rest of this chapter).

Every Reader needs a source port to read from. Such a port could be console input for the REPL, a file on local filesystem, a TCP connection, socket, or just an in-memory string². Reading program source starts with a top-most Reader, which we may call the *Root Reader*, which is implicitly (unless defined otherwise) `Aml/Base.Lang.Reader`.

Readers are required to implement two methods: `read` and `read-syntax`.

The `read` method must accept a source port as its only argument, and it can return any type of value.

The `read-syntax` method must accept a source name argument along with second argument of the source port, and it must return either the `System/IO.Eof` value, or a `System/Lang.Syntax_Object` value.

The Root Reader's `read-syntax` method is used on the top-most initial port.³

¹Some ideas for that would include natural speech interface (at least in English).

²More about strings later.

³Might be different in future, when we allow it to use `read` instead. That is not very useful now though.

2.1 Root Reader Abilities

The Root Reader can recognise basic delimiters, such as all whitespace and newlines.

Delimited by these, it can recognise the following forms:

- Shebang, such as `#!/usr/bin/env aml`, to point to the entry-level Amlantis System tool.
- Reader switch, `#reader name ...`, which specifies a reader to use for the following forms, and switches to it. The *name* should be a simple identifier.⁴
- Language switch, `#lang name ...`, which expands right away into `#reader name.Lang.Reader ...`⁵ The *name* should be a simple identifier.⁶

⁴Specification for identifiers will be provided later.

⁵It may expand to other forms as well in certain order, due to be defined later.

⁶Specification for identifiers will be provided later.

Part III

The Amlantis System Tools

Chapter 3

The Top-Level System or REPL – `aml`

This chapter will describe the top-level system for the Amlantis System, which serves as universal entry point for running programs designed for this system.

Without input files specified in arguments, it permits interactive use of the Amlantis System through a mechanism known as read-eval-print loop (REPL). With this mode, the system repeatedly reads user's input, evaluates it with the context of every previous input of the same REPL session, prints the inferred type of the result of such evaluation along with the text representation of the result, if any, and repeats.

Chapter 4

The Compiler – `amlc`

In this chapter, we're going to describe the Amlantis System Compiler `amlc`, which compiles source code files to bytecode files. The compilation process involves reading through the input source files, expanding them by evaluating the read programs by using the specified language, and serializing the type system's state along with modules' state to bytecode files, which are then to be run by `amlrun` (§5) or `aml` (§3).

Chapter 5

The Runtime System – `amlrun`

This chapter will describe the `amlrun` tool, which reads files output by `amlc` (§4) and executes a specified executable¹.

¹Due to be described later, but those could be e.g. a specific function, or a script file.

Chapter 6

The Debugger – `amldebug`

In this chapter, we'll describe the Amlantis System's own debugger tool.

Chapter 7

The Profiler – amlprof

This chapter will describe the Amlantis System's profiling tool.

Chapter 8

The Builder – `amlbuild`

In this chapter, we'll describe the Amlantis System's builder tool.

Chapter 9

The Book Workspace – amlbook

This chapter will describe the Amlantis System's "Book Workspace" tool, which is a planned addition to the whole toolchain and supposed to be accompanied by a GUI. The purpose will be to work with source texts that intertwine program source code in snippets with rich text documents, and ability to selectively execute these program source codes, seeing results right below each snippet.