Table of Contents

Env Package Documentation

This package provides utilities for interacting with the runtime environment, in cluding access to command line arguments, current time, and working directory in formation.

Command Line Arguments

Access command line arguments passed to your program:

```
1
2   test "command line arguments" {
3     let arguments = @env.args()
4
5
6
7     inspect(arguments.length() >= 0, content="true")
8
9
10    fn process_args(args : Array[String]) -> String {
11      if args.length() == 0 {
12        "No arguments provided"
13      } else {
14        "First argument: " + args[0]
15      }
16    }
17
18    let result = process_args(arguments)
19    inspect(result.length() > 0, content="true")
20  }
```

Current Time

Get the current time in milliseconds since Unix epoch:

```
1
2   test "current time" {
3     let timestamp = @env.now()
4
5
6     let year_2020_ms = 1577836800000UL
7     inspect(timestamp > year_2020_ms, content="true")
8
9
10    fn format_timestamp(ts : UInt64) -> String {
11      "Timestamp: " + ts.to_string()
12    }
13
14    let formatted = format_timestamp(timestamp)
15    inspect(formatted.length() > 10, content="true")
16  }
```

Working Directory

Get the current working directory:

```
1
2   test "working directory" {
3     let cwd = @env.current_dir()
4     match cwd {
5       Some(path) => {
6
7         inspect(path.length() > 0, content="true")
8         inspect(path.length() > 1, content="true")
9       }
10      None =>
11
12        inspect(true, content="true")
13    }
14  }
```

Practical Usage Examples

Command Line Tool Pattern

```
1
2   test "command line tool pattern" {
3     fn parse_command(args : Array[String]) -> Result[String, String] {
4       if args.length() < 2 {
5         Err("Usage: program <command> [args...]")
6       } else {
7         match args[1] {
8           "help" => Ok("Showing help information")
9           "version" => Ok("Version 1.0.0")
10          "status" => Ok("System is running")
11          cmd => Err("Unknown command: " + cmd)
12        }
13      }
14    }
15
16
17    let test_args = ["program", "help"]
18    let result = parse_command(test_args)
19    inspect(result, content="Ok(\"Showing help information\")")
20    let invalid_result = parse_command(["program", "invalid"])
21    match invalid_result {
22      Ok(_) => inspect(false, content="true")
23      Err(msg) => inspect(msg.length() > 10, content="true")
24    }
25  }
```

Configuration Loading

```
1
2   test "configuration loading" {
3     fn load_config_path() -> String {
4       match @env.current_dir() {
5         Some(cwd) => cwd + "/config.json"
6         None => "./config.json"
7       }
8     }
9
10    let config_path = load_config_path()
11    inspect(config_path.length() > 10, content="true")
12  }
```

Logging with Timestamps

```
1
2   test "logging with timestamps" {
3     fn log_message(level : String, message : String) -> String {
4       let timestamp = @env.now()
5       "[" + timestamp.to_string() + "] " + level + ": " + message
6     }
7
8     let log_entry = log_message("INFO", "Application started")
9     inspect(log_entry.length() > 20, content="true")
10    inspect(log_entry.length() > 10, content="true")
11  }
```

File Path Operations

```
1
2   test "file path operations" {
3     fn resolve_relative_path(relative : String) -> String {
4       match @env.current_dir() {
5         Some(base) => base + "/" + relative
6         None => relative
7       }
8     }
9
10    let resolved = resolve_relative_path("data/input.txt")
11    inspect(resolved.length() > 10, content="true")
12  }
```

Platform Differences

The env package behaves differently across platforms:

JavaScript Environment

- args() returns arguments from the JavaScript environment
- @env.now() uses Date.@env.now()
- @env.current_dir() may return None in browser environments

WebAssembly Environment

- args() behavior depends on the WASM host
- @env.now() provides millisecond precision timing
- @env.current_dir() availability depends on host capabilities

Native Environment

- args() returns actual command line arguments
- @env.now() provides system time
- @env.current_dir() uses system calls to get working directory

Error Handling

Handle cases where environment information is unavailable:

```
1
2   test "error handling" {
3     fn safe_get_cwd() -> String {
4       match @env.current_dir() {
5         Some(path) => path
6         None =>
7
8           "."
9       }
10    }
11
12    let safe_cwd = safe_get_cwd()
13    inspect(safe_cwd.length() > 0, content="true")
14    fn validate_args(
15      args : Array[String],
16      min_count : Int,
17    ) -> Result[Unit, String] {
18      if args.length() < min_count {
19        Err("Insufficient arguments: expected at least " + min_count.to_st
20      } else {
21        Ok(())
22      }
23    }
24
25    let validation = validate_args(["prog"], 2)
26    match validation {
27      Ok(_) => inspect(false, content="true")
28      Err(msg) => inspect(msg.length() > 10, content="true")
29    }
30  }
```

Best Practices

1. Handle Missing Environment Data Gracefully

```

  test "graceful handling" {
    fn get_work_dir() -> String {
      match @env.current_dir() {
        Some(dir) => dir
        None => "~"
      }
    }

    let work_dir = get_work_dir()
    inspect(work_dir.length() > 0, content="true")
  }
```

2. Validate Command Line Arguments

```

  test "argument validation" {
    fn validate_and_parse_args(
      args : Array[String],
    ) -> Result[(String, Array[String]), String] {
      if args.length() == 0 {
        Err("No program name available")
      } else if args.length() == 1 {
        Ok((args[0], []))
      } else {
        let program = args[0]
        let arguments = Array::new()
        for i in 1..<args.length() {
          arguments.push(args[i])
        }
        Ok((program, arguments))
      }
    }

    let test_result = validate_and_parse_args(["myprogram", "arg1", "arg2"
    match test_result {
      Ok((prog, args)) => {
        inspect(prog, content="myprogram")
        inspect(args.length(), content="2")
      }
      Err(_) => inspect(false, content="true")
    }
  }
```

3. Use Timestamps for Unique Identifiers

```
1
2   test "unique identifiers" {
3     fn generate_unique_id(prefix : String) -> String {
4       prefix + "_" + @env.now().to_string()
5     }
6
7     let id1 = generate_unique_id("task")
8     let id2 = generate_unique_id("task")
9     inspect(id1.length() > 10, content="true")
10    inspect(id2.length() > 10, content="true")
11
12  }
```

Common Use Cases

- **Command Line Tools**: Parse arguments and provide help/usage information
- **Configuration Management**: Load config files relative to current directory
- **Logging Systems**: Add timestamps to log entries
- **File Processing**: Resolve relative file paths
- **Debugging**: Include environment information in error reports
- **Build Tools**: Determine working directory for relative path operations

Performance Considerations

- args() is typically called once at program startup
- @env.now() is lightweight but avoid calling in tight loops if high precision i sn’t needed
- @env.current_dir() may involve system calls, so cache the result if used frequ ently
- Environment functions are generally fast but platform-dependent

The env package provides essential runtime environment access for building robus t MoonBit applications that interact with their execution environment.