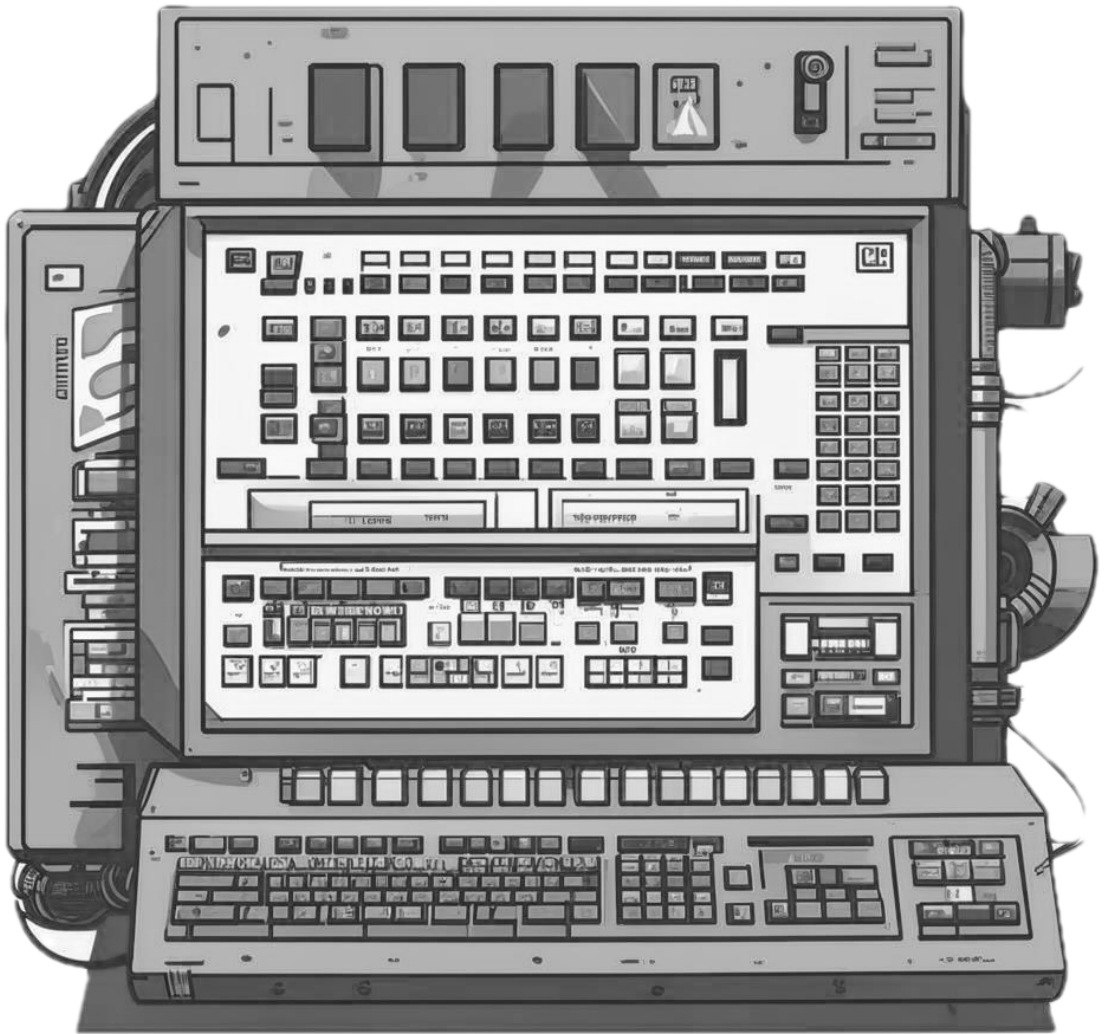




Composed by Vladimir Ulogov

BUND standard library reference

This book serves as a reference guide for the BUND functions (or “words”) defined in standard library.

Referencing [93](#) functions.

I want to thank my first teacher, who imparted the knowledge and guidance necessary to develop my first programs
for the PDP-11 computer.

Introduction

I will introduce a new concatenative programming language called BUND in this work. What is a concatenative language, and how does it differ from the programming languages you're likely familiar with? You're likely acquainted with applicative programming languages like Python, C, or Java. Alternatively, you may have discovered functional programming languages such as Lisp, Haskell, or ML, other examples of applicative programming languages. This category is defined by the way functions are viewed and handled. In applicative languages, a function is treated as a mathematical primitive that computes based on passed arguments and returns a value. In contrast, concatenative programming languages pass a data context from one function to another, external to the function itself. While the stack is the most common method for passing such context, there are concatenative languages that don't utilize a stack. Passing data context enables the concatenation of data processing. Concatenative languages are less known in the software development communities, but you might have heard of languages such as Forth, PostScript, and Factor.

The stack is utilized in many but not all concatenative languages, while applicative languages often use stack structures internally to aid computation. Stacks are indispensable for recursive computation, passing return values computed by functions and storing references to an execution context. What distinguishes concatenative stack-based languages from applicative counterparts is the use of the stack for input data, computational context, and result storage. In essence, everything in concatenative stack-based languages is stored in the stack. In some cases, computational instructions are also stored alongside data on the stack. Since everything, including the context for functions, is stored on

the stack, functions in concatenative stack-based languages do not have conventional arguments. Although they function as such, they are often referred to as “words,” as was defined in one of the first concatenative languages to gain popularity - Forth. Another characteristic of concatenative stack-based languages is their reliance on the stack’s Last In, First Out (LIFO) nature. They often employ Reverse Polish Notation (RPN).

So, what will might surprise you in concatenative stack-based language?

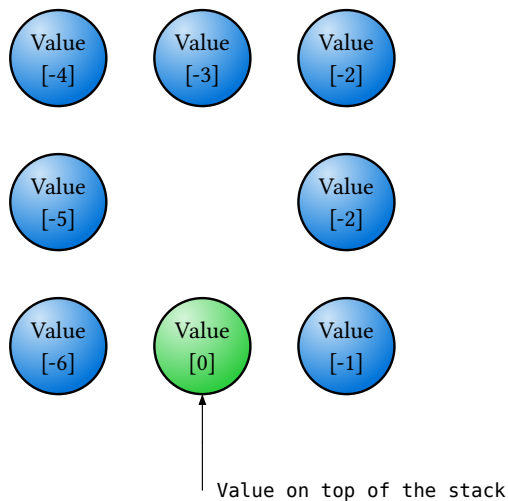
- We already mentioned that the functions do not have arguments and no dedicated return value. All input and output data passed to and from the function are passed through the stack.
- You are responsible for ensuring the correct order of the values passed in the data context to the function, as this context is on the stack.
- You are also responsible for interpreting return data placed on the stack. Unlike in the functional language paradigm, there could be more than one return value, depending on your function (or “word”).
- There are no variables. All data are stored on the stack.
- There are no global constants, variables, or values. Everything is on the stack.
- Due to the LIFO nature of the stack, you will deal with RPN, although BUND offers you an ability to create a stack with FIFO policy.

What exactly is a Bund?

The BUND programming language is a member of the concatenative language family. A notable characteristic of concatenative languages is the presence of a computational context external to the code itself. All computations carried out by the functions, referred to as “words” in concatenative language terminology, are performed over this external context. This differs from the concepts commonly encountered in applicative languages, where function parameters are part of the function context. The computational context is typically structured as a Last In, First Out (LIFO) stack in concatenative languages. However, BUND distinguishes itself from most concatenative languages by having a more sophisticated concept of the computational context.

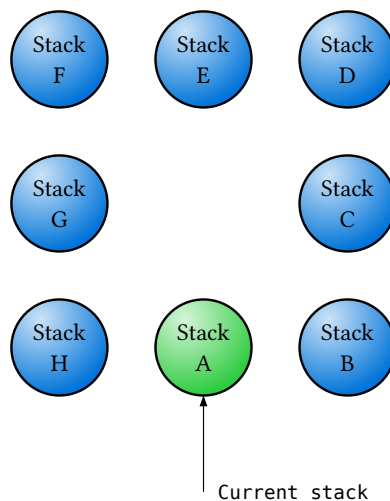
Circular data stack

Instead of using simple LIFO stacks, BUND stores data in multiple named circular buffers, also known as stacks. When you push data to the stack, the circular buffer expands, and when you pull or consume data from the stack, the buffer contracts. While the data buffer is circular, there is always a pointer that refers to the value located on top of the stack. Although you can rotate the buffer in the left or right direction, data is consumed in a single direction only.



Stack-of-stacks references

The next level of abstraction is a circular stack that refers to named data stacks while functioning just like a standard data stack in all other aspects. The stack referred to by the “top of the stack” reference is considered the “current stack,” and all operations are by default performed within this data context. When creating a new stack, the reference moves to the top of the stack. When positioning a named stack to become the current stack, the buffer rotates to bring the required stack to the proper position at the “top of the stack.”



Workbench

The workbench, an integral component of the BUND virtual machine, is a circular stack that temporarily holds and transfers values between computations conducted in various data contexts. Despite its functional significance, this circular stack does not carry a specific name.

What does the word “bund” mean?

The term “*Bund*”¹ comes from German or Yiddish and can be translated as “*association*,” “*bundle*,” or “*bunch*.” Throughout history, this word has been utilized in various contexts. In the context of the multi-stack concatenative programming language, “Bund” refers to the ability of the BUND language to integrate distinct, originally separate data and computation contexts into a unified computational process aimed at achieving a common goal.

¹singular *Bundes*, plural *Bunde*

How to use the reference?

Generally, the reference does not require details on how you present information about your topic, but I still feel obligated to explain the structure of the function reference page.

- ☐ At the top of the page, you will find an optional warning indicating that using this function requires additional caution from the developer.
- ☐ Next, a list will provide details about where the function is implemented.
- ☐ Following that, there is a description of the operations performed by the function.
- ☐ Afterward, you will see a concise outline of the function's algorithm, highlighting how it interacts with the stack or workbench, including its inputs and outputs.
- ☐ Finally, a code snippet will demonstrate how to utilize the function effectively.

Clear current stack - *clear*

Name of the function

 Danger

This is destructive operation with stack.

Warning, if any

Defined in

Where the function is implemented

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

This function clears out all data from the current stack

Description

```

1: function CLEAR()
2:   ▷ Clearing all values in current stack
3:   Name ← VM::current_stack_name()
4:   if Name = None then
5:     return Error("Error getting current stack name")
6:   CLEAR_STACK(Name)

```

```
//
// Clearing current stack
//
clear
```

Function algorithm

Code sample

Figure 1: Here is your guide to the common Standard Library reference page.

BUND Standard library reference

Although language design is often simple, elegant, and thoughtfully executed, there is room for greater practicality. A language's core becomes truly functional and valuable to developers only when accompanied by a standard library of useful functions. These functions provide essential tools for performing operations and manipulating data effectively. Moreover, BUND shares several characteristics with other concatenative languages.

! Memorize

All run-time functionality of the BUND implemented in standard library.

When I say “all,” I mean that every aspect of the functionality extends beyond just implementing the BUND parser and core logic. The BUND standard library is situated across multiple locations. Although this may initially be a design flaw, I had deliberate reasons for structuring the standard library this way.

i Info

- ☐ The Rust crate *rust_multistack* encompasses all the logic associated with stack operations. Additionally, it incorporates elements of the standard library that pertain specifically to these operations. Features include data swapping on the stack, data duplication, removal of data, stack rotation, creation of stacks, and other related functionalities.
- ☐ The Rust crate *rust_multistackvm* is a foundational implementation of the BUND virtual machine. Although different tools and interpreters may facilitate access to BUND, the core logic of the BUND language remains intact within this crate. This encompasses data manipulation and conversion, application logic, mathematical operations, lambda function processing, and all other essential features.
- ☐ The Bund runtime serves as the interpreter for the Bund programming language. It encompasses implementing all standard library functions and facilitating user-related features and controls.

Execute code snippet in the BUND debugger - *debug*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Execute code snippet inside BUND debugger

```
1: function DEBUG_SHELL()
2:   ▷ Bund debugger
3:   Snippet ← current stack
4:   if Snippet = None then
5:     return Error("Stack is too shallow")
6:   BUND_DEBUGGER(Snippet)
```

```
//
"2 2 + println" debug
```

1
2

Arguments passed via CLI for script and shell subcommands - *args*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Users can pass arguments to the script and shell subcommands after `--`. For example, to pass arguments to the shell subcommand, you would use:

```
bund shell -- "Argument1" 2 3
```

The “word” `args` will return to the top of the stack list containing the passed arguments.

- 1: **function** FUNCTION-NAME()
- 2: ▷ Return passed arguments (if any)
- 3: *current stack* ← ARGS

```
// 1
// Printing passed arguments 2
// bund sctipt --stdin -- 1 2 3 3
// 4
args println 5
// Will output [1:: 2:: 3] 6
```

Parsing arguments passed via CLI for script and shell subcommands - *args.parse*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Parsing passed arguments and returning dictionary containing parsed arguments to the stack.

- 1: **function** FUNCTION-NAME()
- 2: ▷ Return parsed arguments (if any)
- 3: *current stack* ← Call(“Args.Parse”, [ARGS])

```
//  
// Printing passed and parsed arguments  
//  
args.parse println
```

1
2
3
4

Load and execute scripts stored as bootstrap scripts in the WORLD file - *bootstrap*

Danger

Scripts executed from the external file will alter current VM state

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

With the help of the *save* function, users can store the current state of the BUND VM into an external file. This file, called the “WORLD” file, contains a frozen version of the scripts that could be executed during bootstrap phase or by calling a *bootstrap* function. The *bootstrap* function, which can be used as `"WORLD_file"` bootstrap, will load content from this file and execute stored bootstrap scripts.

```
1: function BUND-BOOTSTRAP()
2:     ▷ Load and execute bootstrap scripts from the WORLD file
3:     Filename ← current stack
4:     if Value = None then
5:         return Error("Stack is too shallow")
6:     BUND_BOOTSTRAP(Filename)
```

```
// Load and execute scripts from file _state.world_      1
"state" bootstrap                                         2
```

Evaluate BUND code snippet taken as string from stack - *bund.eval*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

This function is tasked with the evaluation of a code snippet that is retrieved from a string stored in the stack. The process involves standard error-handling mechanisms to manage any issues that may arise during execution. It is crucial to highlight that this function does not instantiate a new virtual machine.“

```
1: function BUND-EVAL()
2:     ▷ Evaluate bund code snippet
3:     Snippet ← current stack
4:     if Snippet = None then
5:         return Error(“Stack is too shallow”)
6:     BUND_EVAL(Snippet)
```

```
// 1
// Evaluate code snippet from the sting stored in the 2
stack 3
// 4
"2 2 +" bund.eval 5
// Leaves 42 on the stack
```

Evaluate BUND code snippet taken from the file. The name of the file is taken as string from stack - *bund.eval-file*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

This function is responsible for the evaluation of a code snippet that is extracted from a string loaded from a specified file. The name of this file is derived from a string that is stored in the stack . All errors encountered during the execution of this function are managed using standard error-handling procedures. It is important to note that this function does not instantiate a new virtual machine.

```
1: function BUND-EVAL-FILE()
2:   ▷ Evaluate bund code snippet
3:   Snippet ← Call("Read", [current stack])
4:   if Snippet = None then
5:     return Error("Stack is too shallow")
6:   BUND_EVAL(Snippet)
```

```
//
// Evaluate code snippet stored in the file
//
"helloworld.bund" bund.eval-file
// Prints "Hello World!"
```

1
2
3
4
5

Evaluate BUND code snippet taken from the file. The name of the file is taken as string from workbench - *bund.eval-file*.

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

This function is responsible for the evaluation of a code snippet that is extracted from a string loaded from a specified file. The name of this file is derived from a string that is stored within the workbench environment. All errors encountered during the execution of this function are managed using standard error-handling procedures. It is important to note that this function does not instantiate a new virtual machine.

```
1: function BUND-EVAL-FILE-DOT()
2:   ▷ Evaluate bund code snippet
3:   Snippet ← Call("Read", [workbench])
4:   if Snippet = None then
5:     return Error("Workbench is too shallow")
6:   BUND_EVAL(Snippet)
```

```
// 1
// Evaluate code snippet stored in the file 2
// 3
"hellworld.bund" . bund.eval-file. 4
// Prints "Hello World!" 5
```

Evaluate BUND code snippet taken as string from workbench - *bund.eval*.

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

This function is tasked with the evaluation of a code snippet that is retrieved from a string stored in the workbench. The process involves standard error-handling mechanisms to manage any issues that may arise during execution. It is crucial to highlight that this function does not instantiate a new virtual machine.“

```
1: function BUND-EVAL-FROM-WORKBENCH()
2:     ▷ Evaluate bund code snippet
3:     Snippet ← workbench
4:     if Snippet = None then
5:         return Error(“Workbench is too shallow”)
6:     BUND_EVAL(Snippet)
```

```
// 1
// Evaluate code snippet from the sting 2
// dynamically created and stored in the workbench 3
// 4
"2 " "2 +" + . bund.eval. 5
// Leaves 42 on the stack 6
```

Terminate BUND interpreter and exit - *bund.exit*

Danger

BUND interpreter will be terminated

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Terminate BUND interpreter, exit and return an exit code to OS

Aliases

- ☐ exit

```
1: function FUNCTION-NAME()
2:     ▷ Terminate BUND interpreter
3:     Exit_code ← current stack
4:     if Value = None then
5:         EXIT(0)
6:     EXIT(Exit_code)
```

```
// 1
// Exit with code 10 2
// 3
10 exit 4
```

Clear current stack - *clear*

Danger

This is destructive operation with stack.

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

This function clears out all data from the current stack

```
1: function CLEAR()
2:   ▷ Clearing all values in current stack
3:   Name ← VM::current_stack_name()
4:   if Name = None then
5:     return Error("Error getting current stack name")
6:   CLEAR_STACK(Name)
```

```
// 1
// Clearing current stack 2
// 3
1 2 3 clear 4
// After calling clear stack is going to be empty 5
```

Clear named stack - *clear_in*

Danger

This is destructive operation with stack.

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

This function clears out all data from the named stack, taking the name of the stack from current stack

```
1: function CLEAR_IN()
2:     ▷ Clearing named stack
3:     Name ← current stack
4:     if Name = None then
5:         return Error("Stack is too shallow")
6:     CLEAR_STACK(Name)
```

```
// 1
// Remove all data from named stack 2
// 3
@main 4
@StackName 5
  1 2 3 6
@main 7
  :StackName clear_in 8
// After calling clear_in, stack with name "StackName" 9
// will have no data 10
```

Taking value from stack and convert it to boolean - *convert.to_bool*

Defined in

- ☐ rust_multistack
- ☒ rust_multistackvm
- ☐ bund runtime

Taking value from the stack, converting to the BOOL and pushing result to the stack

```
1: function CONVERT_To_BOOL()
2:   ▷ Converting Value to Boolean
3:   Value ← current stack
4:   if Value = None then
5:     return Error("Stack is too shallow")
6:   current stack ← Value::conv(BOOL)
```

```
// 1
// Converting data in stack to string 2
// 3
:TRUE convert.to_bool 4
TRUE == { 5
  "Conversion is succesful" 6
  println 7
} if 8
```

Taking value from workbench and convert it to boolean - *convert.to_bool*.

Defined in

- ☐ rust_multistack
- ☒ rust_multistackvm
- ☐ bund runtime

Taking value from the workbench, converting to the BOOLEAN and pushing result to the workbench

```
1: function CONVERT_To_BOOL_IN_WORKBENCH()
2:   ▷ Converting Value to Boolean
3:   Value ← workbench
4:   if Value = None then
5:     return Error("Workbench is too shallow")
6:   workbench ← Value::conv(BOOL)
```

```
1 . convert.to_bool. take
TRUE == {
  "Conversion is succesful"
  println
} if
```

1
2
3
4
5

Taking value from stack and convert it to float - *convert.to_float*

Defined in

- ☐ rust_multistack
- ☒ rust_multistackvm
- ☐ bund runtime

Taking value from the stack, converting to the FLOAT and pushing result to the stack

```
1: function CONVERT_To_FLOAT()
2:   ▷ Converting Value to Float
3:   Value ← current stack
4:   if Value = None then
5:     return Error("Stack is too shallow")
6:   current stack ← Value::conv(FLOAT)
```

```
// 1
// Converting data in stack to string 2
// 3
42 convert.to_float 4
42.0 == { 5
  "Conversion is succesful" 6
  println 7
} if 8
```

Taking value from workbench and convert it to float - *convert.to_float*.

Defined in

- ☐ rust_multistack
- ☒ rust_multistackvm
- ☐ bund runtime

Taking value from the workbench, converting to the FLOAT and pushing result to the workbench

```
1: function CONVERT_To_FLOAT_IN_WORKBENCH()
2:   ▷ Converting Value to String
3:   Value ← workbench
4:   if Value = None then
5:     return Error("Workbench is too shallow")
6:   workbench ← Value::conv(FLOAT)
```

```
42 . convert.to_string. take
42.0 == {
  "Conversion is succesful"
  println
} if
```

1
2
3
4
5

Taking value from stack and convert it to int - *convert.to_int*

Defined in

- ☐ rust_multistack
- ☒ rust_multistackvm
- ☐ bund runtime

Taking value from the stack, converting to the INT and pushing result to the stack

```
1: function CONVERT_To_INTEGER()
2:   ▷ Converting Value to Integer
3:   Value ← current stack
4:   if Value = None then
5:     return Error("Stack is too shallow")
6:   current stack ← Value::conv(INT)
```

```
// 1
// Converting data in stack to int 2
// 3
TRUE convert.to_int 4
1 == { 5
  "Conversion is succesful" 6
  println 7
} if 8
```

Taking value from workbench and convert it to int - *convert.to_int*.

Defined in

- ☐ rust_multistack
- ☒ rust_multistackvm
- ☐ bund runtime

Taking value from the workbench, converting to the INT and pushing result to the workbench

```
1: function CONVERT_To_INTEGER_IN_WORKBENCH()
2:   ▷ Converting Value to Integer
3:   Value ← workbench
4:   if Value = None then
5:     return Error("Workbench is too shallow")
6:   workbench ← Value::conv(INT)
```

```
42.0 . convert.to_int. take
42 == {
  "Conversion is succesful"
  println
} if
```

1
2
3
4
5

Taking value from stack and convert it to list - *convert.to_list*

Defined in

- ☐ rust_multistack
- ☒ rust_multistackvm
- ☐ bund runtime

Taking value from the stack, converting to the LIST and pushing result to the stack

```
1: function CONVERT_To_LIST()
2:   ▷ Converting Value to List
3:   Value ← current stack
4:   if Value = None then
5:     return Error("Stack is too shallow")
6:   current stack ← Value::conv(LIST)
```

```
// 1
// Converting data in stack to list 2
// 3
pair : 4
  41 42 5
; convert.to_list 6
[ 41 42 ] == { 7
  "Conversion is succesful" 8
  println 9
} if 10
```

Taking value from workbench and convert it to list - *convert.to_list*.

Defined in

- ☐ rust_multistack
- ☒ rust_multistackvm
- ☐ bund runtime

Taking value from the workbench, converting to the LIST and pushing result to the workbench

```
1: function CONVERT_To_LIST_IN_WORKBENCH()
2:   ▷ Converting Value to List
3:   Value ← workbench
4:   if Value = None then
5:     return Error("Workbench is too shallow")
6:   workbench ← Value::conv(STRING)
```

```
pair :
  41 42
; . convert.to_list.
[ 41 42 ] ==. {
  "Conversion is succesful"
  println
} if
```

1
2
3
4
5
6
7

Taking value from stack and convert it to matrix - *convert.to_matrix*

Defined in

- ☐ rust_multistack
- ☒ rust_multistackvm
- ☐ bund runtime

Taking value from the stack, converting to the MATRIX and pushing result to the stack

```
1: function CONVERT_To_MATRIX()
2:   ▷ Converting Value to Matrix
3:   Value ← current stack
4:   if Value = None then
5:     return Error("Stack is too shallow")
6:   current stack ← Value::conv(MATRIX)
```

```
// 1
// Converting data in stack to matrix 2
// 3
[ 4
  [ 1 2 3 ] 5
  [ 4 5 6 ] 6
] convert.to_matrix dup 7
"Size of matrix " print len println 8
```

Taking value from workbench and convert it to matrix - *convert.to_matrix*.

Defined in

- ☐ rust_multistack
- ☒ rust_multistackvm
- ☐ bund runtime

Taking value from the workbench, converting to the MATRIX and pushing result to the workbench

```
1: function CONVERT_To_MATRIX_IN_WORKBENCH()
2:   ▷ Converting Value to Matrix
3:   Value ← workbench
4:   if Value = None then
5:     return Error("Workbench is too shallow")
6:   workbench ← Value::conv(MATRIX)
```

```
// 1
// Summing two matrixes, one is in workbench 2
// 3
[ 4
  [ 1 2 3 ] 5
  [ 4 5 6 ] 6
] . convert.to_matrix. 7
[ 8
  [ 7 8 9 ] 9
  [ 10 11 12 ] 10
] convert.to_matrix +. println 11
```

Taking value from stack and convert it to string - *convert.to_string*

Defined in

- ☐ rust_multistack
- ☒ rust_multistackvm
- ☐ bund runtime

Taking value from the stack, converting to the STRING and pushing result to the stack

```
1: function CONVERT_To_STRING()
2:   ▷ Converting Value to String
3:   Value ← current stack
4:   if Value = None then
5:     return Error("Stack is too shallow")
6:   current stack ← Value::conv(STRING)
```

```
// 1
// Converting data in stack to string 2
// 3
42 convert.to_string 4
"42" == { 5
  "Conversion is succesful" 6
  println 7
} if 8
```

Taking value from workbench and convert it to string - *convert.to_string*.

Defined in

- ☐ rust_multistack
- ☒ rust_multistackvm
- ☐ bund runtime

Taking value from the workbench, converting to the STRING and pushing result to the workbench

```
1: function CONVERT_To_STRING_IN_WORKBENCH()
2:   ▷ Converting Value to String
3:   Value ← workbench
4:   if Value = None then
5:     return Error("Workbench is too shallow")
6:   workbench ← Value::conv(STRING)
```

```
42 . convert.to_string.
"42" ==. {
  "Conversion is succesful"
  println
} if
```

1
2
3
4
5

Taking value from stack and convert it to textbuffer - *convert.to_textbuffer*

Defined in

- ☐ rust_multistack
- ☒ rust_multistackvm
- ☐ bund runtime

Taking value from the stack, converting to the TEXTBUFFER and pushing result to the stack

```
1: function CONVERT_To_TEXTBUFFER()
2:   ▷ Converting Value to Textbuffer
3:   Value ← current stack
4:   if Value = None then
5:     return Error("Stack is too shallow")
6:   current stack ← Value::conv(TEXTBUFFER)
```

```
//
// Converting data in stack to textbuffer
//
"Hello"
  convert.to_textbuffer
  "WSorld" ,
  "!" ,
println
// And printing Hello World!
```

1
2
3
4
5
6
7
8
9

Taking value from workbench and convert it to textbuffer - *convert.to_textbuffer*.

Defined in

- ☐ rust_multistack
- ☒ rust_multistackvm
- ☐ bund runtime

Taking value from the workbench, converting to the TEXTBUFFER and pushing result to the workbench

```
1: function CONVERT_To_TEXTBUFFER_IN_WORKBENCH()
2:   ▷ Converting Value to Textbuffer
3:   Value ← workbench
4:   if Value = None then
5:     return Error("Workbench is too shallow")
6:   workbench ← Value::conv(TEXTBUFFER)
```

```
// Creating message in TEXTBUFFER      1
"Hello" .                               2
  convert.to_textbuffer.                3
  "World" +.                            4
take println                            5
// And printing Hello World!           6
```

Return name of current stack to current stack - *current*

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Returns the name of current stack to current stack

```
1: function CURRENT()
2:     ▷ Returns the name of current stack to stack
3:     Name ← VM::current_stack_name()
4:     if Name = None then
5:         return Error("Can not detect current stack name")
6:     current stack ← Name
```

```
// 1
// Prints the name of current stack 2
// 3
current println 4
```

Display information about host - *debug.display_hostinfo*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Display information about current host on which you are executing BUND code

```
1: function DEBUG_DISPLAY_HOSTINFO()  
2:     ▷ Display current Host information  
3:     DEBUG_DISPLAY_HOSTINFO()
```

```
//  
debug.display_hostinfo
```

1
2

Display data stored in current stack - *debug.display_stack*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Display the data stored into the current stack to the standard output. This debug command doesn't change stack state and stored values.

```
1: function DEBUG_DISPLAY_STACK()
2:     ▷ Display stack state
3:     DEBUG_DISPLAY_STACK()
```

```
// 1
debug.display_stack 2
```

Display data stored in workbench - *debug.display_workbench*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Display the data stored into the workbench to the standard output. This debug command doesn't change workbench state and stored values.

```
1: function DEBUG_DISPLAY_WORKBENCH()  
2:     ▷ Display Workbench State  
3:     DEBUG_DISPLAY_WORKBENCH()
```

```
//  
debug.display_workbench
```

1
2

Drop into a debug shell - *debug.shell*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Force to drop to the debug shell of the BUND interpreter

```
1: function DEBUG_SHELL()  
2:     ▷ Drop into a debug shell  
3:     DEBUG_SHELL()
```

```
// 1  
debug.shell 2  
// You can exit from debug.shell by pressing Ctrl-D 3
```

Decoding string from stack from BASE64 - *decode.base64*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Taking data in BASE64 from the stack and decoding it. Return decoded data to the stack.

```
1: function DECODE-BASE64()
2:   ▷ Decoding from BASE64
3:   Data ← current stack
4:   if Data = None then
5:     return Error("Stack is too shallow")
6:   current stack ← Call("Decode_Base64", [Data])
```

```
// 1
// Encode and Decode data in BASE64 2
// 3
"Hello world!" encode.base64 decode.base64 println 4
// Prints "Hello world!" 5
```

Decoding string from workbench from BASE64 - *decode.base64*.

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Taking data in BASE64 from workbench and decoding it. Return decoded data to the workbench.

```
1: function DECODE-BASE64-WORKBENCH()  
2:   ▷ Decoding from BASE64  
3:   Data ← workbench  
4:   if Data = None then  
5:     return Error("Workbench is too shallow")  
6:   workbench ← Call("Decode_Base64", [Data])
```

```
// 1  
// Encode and Decode data in BASE64 2  
// 3  
"Hello world!" . encode.base64. decode.base64. take 4  
println  
// Prints "Hello world!" 5
```

Drop element from the stack - *drop*

Danger

This is destructive operation with data.

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

This function takes a single value from the top of current stack and discards it.

```
1: function DROP()
2:   ▷ Dropping value that is on top of the stack
3:   Name ← VM::current_stack_name()
4:   if Value = None then
5:     return Error("Stack is too shallow")
6:   DROP(Name)
```

```
// 1
// Calling this function will remove 2
// and discard a value 3
// 4
42 drop 5
```

Drop the last value in the named stack - *drop_in*

Danger

This is destructive operation with data.

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Drop the last value in the named stack

```
1: function DROP_IN()
2:   ▷ Drop the value in the named stack
3:   Name ← VM::current_stack_name()
4:   if Name = None then
5:     return Error("Stack is too shallow")
6:   DROP(Name)
```

```
// 1
// Drop the last value from stack "A" 2
// 3
"A" drop_in 4
```

Remove stack with all data - *drop_stack*

Danger

This is destructive operation with stack.

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Drop named stack.

```
1: function DROP_STACK()
2:     ▷ Drop the stack
3:     Name ← current stack
4:     if Value = None then
5:         return Error("Stack is too shallow")
6:     DROPSTACK(Name)
```

```
// 1
// Drop stack "TheStack" 2
// 3
:TheStack drop_stack 4
// Now stack _TheStack_ doesn't exists 5
```

Duplicate multiple values in the current stack - *dup_many*

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Duplicate multiple values in current stack

```
1: function DUP_MANY()
2:   ▷ Duplicate multiple values
3:   N ← current stack
4:   Name ← VM::current_stack_name()
5:   while N >= 0 do
6:     Value ← current stack
7:     current stack ← Call("Dup", [N, Name, Value])
8:     N ← N - 1
```

```
// 1
// Duplicate data in stack 2
// 3
42 41 2 dup_many 4
// Now we have 42, 42, 41, 41 in stack 5
```

Duplicate multiple values in the named stack - *dup_many_in*

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Duplicating multiple items in the named stack

```
1: function DUP_MANY_IN()
2:     ▷ Duplicate multiple items in the named stack
3:     Name ← current stack
4:     Name ← current stack
5:     if Name = None then
6:         return Error("Stack is too shallow")
7:     if N = None then
8:         return Error("Stack is too shallow")
9:     while N >= 0 do
10:        Value ← Name stack
11:        Name stack ← Call("Dup", [N, Name, Value])
12:        N ← N - 1
```

```
@main 1
@StackName 2
1 2 3 3
@main 4
:StackName 3 dup_many_in 5
// Duplicate all three values in 6
// stack "StackName" 7
```

Duplicate single value in the current stack - *dup_one*

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Duplicate single value in current stack

```
1: function DUP_ONE()
2:   ▷ Duplicate value
3:   Value ← current stack
4:   Name ← VM::current_stack_name()
5:   current stack ← Call("Dup", [1, Name, Value])
```

```
// 1
// Duplicate data in stack 2
// 3
42 dup_one 4
// Now we have 42, 42 in stack 5
```

Duplicate single value in the named stack - *dup_one_in*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Description of function

Duplicate single value in the named stack, when name of the stack is reading from current stack

```
1: function DUP_ONE_IN()
2:   ▷ Duplicate value
3:   Name ← current stack
4:   Value ← current stack
5:   stack Name ← Call("Dup", [1, Name, Value])
```

```
// 1
// Duplicating single value from stack "A" 2
// 3
@A 42 4
@main 5
:A dup_one_in 6
// Now in stack A we have 42, 42 7
```

Encoding string from stack into BASE64 - *encode.base64*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Taking data from the stack and encode it into BASE64

```
1: function ENCODE-BASE64()
2:   ▷ Encoding to BASE64
3:   Data ← current stack
4:   if Data = None then
5:     return Error("Stack is too shallow")
6:   current stack ← Call("Encode_Base64", [Data])
```

```
// 1
// Encoding string "Hello world!" to BASE64 2
// 3
"Hello world!" encode.base64 4
// Returns to stack "FQAAAAAAAAABJRy1UdDF1YXV1RUxUcjF50VA5V1FQA0Dn6XT950" 5
```

Encoding string from workbench into BASE64 - *encode.base64*.

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Taking data from workbench and encode it into BASE64. Put result to workbench

```
1: function ENCODE-BASE64()
2:   ▷ Encoding to BASE64
3:   Data ← workbench
4:   if Data = None then
5:     return Error("Stack is too shallow")
6:   workbench ← Call("Encode_Base64", [Data])
```

```
// 1
// Encoding string "Hello world!" to BASE64 2
// 3
"Hello world!" . encode.base64. take 4
// Returns to stack "FQAAAAAAAAABJRy1UdDF1YXV1RUxUcjF50VA5V1tQA0Dn6XT950"
```

Make named stack current, create if stack doesn't exists - *ensure_stack*

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Make named stack current, create if stack does not exists

```
1: function ENSURE_STACK()
2:   ▷ Make stack current, create if not exists
3:   Name ← current stack
4:   if Name = None then
5:     return Error("Stack is too shallow")
6:   if Not Call(Stack_Exists, [Name]) then
7:     CREATE_STACK(Name)
8:   To_STACK(Name)
```

```
//
// Set stack "StackName" current
//
"StackName" ensure_stack
```

1
2
3
4

Make named stack with set capacity current, create if stack doesn't exists. - *ensure_stack_with_capacity*

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Make named stack current, create if stack does not exists with defined capacity

```
1: function ENSURE_STACK_WITH_CAPACITY()
2:   ▷ Make stack current, create if not exists
3:   Name ← current stack
4:   N ← current stack
5:   if Name = None then
6:     return Error("Stack is too shallow")
7:   if Not Call(Stack_Exists, [Name]) then
8:     CREATE_STACK_WITH_CAPACITY(Name, N)
9:   To_STACK(Name)
```

```
// 1
// Set stack "StackName" current 2
// and set stack capacity to not more than 3
// 128 values 4
// 5
128 "StackName" ensure_stack 6
```

Writing string value of the data loaded from the stack into file - *file.write*

Danger

External file will be overwritten

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Writing value representation converted to string to the external file.

```
1: function FUNCTION-NAME()
2:   ▷ Writing string representation of the data to the file
3:   Filename ← current stack
4:   if Filename = None then
5:     return Error("Stack is too shallow")
6:   Data ← current stack
7:   if Data = None then
8:     return Error("Stack is too shallow")
9:   FILE_WRITE(Filename, Call("To_String", [Data]))
```

```
//
// Write "42" to the file fortytwo.txt
//
42 convert.to_string
   :fortytwo.txt
   file.write
```

1
2
3
4
5
6

Folding all values in the current stack - *fold*

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

The *fold* function is designed to extract either all values stored in the stack or just the values that precede the *nodata* entry in the current stack. These extracted values will be compiled into a list and added back to the current stack.

```
1: function FOLD()
2:   ▷ Folding data in the current stack
3:   Result ← []
4:   for Current Stack not Empty do
5:     Value ← current stack
6:     if Value = NODATA then
7:       BREAK()
8:     Result ← Value
9:   current stack ← Result
```

```
// 1
// Folding stack values into list 2
// 3
1 2 3 none 4 5 6 fold 4
// Calling fold will leave 5
// 1 2 3 and [ 4 5 6 ] 6
// in the current stack 7
```

Folding all values in the named stack - *fold_stack*

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

The *fold_stack* function is designed to extract either all values stored in the named stack or just the values that precede the *nodata* entry again, in the named stack. These extracted values will be compiled into a list and added back to the named stack.

```
1: function FOLD()
2:     ▷ Folding data in the named stack
3:     Result ← []
4:     Name ← current stack
5:     for Name not Empty do
6:         Value ← Name stack
7:         if Value = NODATA then
8:             BREAK()
9:         Result ← Value
10:    Name stack ← Result
```

```
// 1
// Folding data in the stack with name "A" 2
@main 3
@A 4
  1 2 3 nodata 4 5 6 5
@main 6
  :A fold_stack 7
// Result of the executing of this 8
// function will be 1 2 3 [4 5 6] 9
// in the stack "A" 10
```

Returning path of current work directory - *fs.cwd*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Return current work directory to the stack

Aliases

- ☐ cwd

```
1: function FS-CWD()  
2:   ▷ Return CWD  
3:   current stack ← Call("Fs_Cwd", [])
```

```
//  
// Prints current work directory  
//  
cwd println
```

1
2
3
4

Generate random v4 UUID - *id.uuid*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Generate random v4 uuid. Result is stored into stack

```
1: function ID.UUID()  
2:   ▷ Generate UUID  
3:   current_stack ← Call("Uuid.new_v4", [])
```

```
// 1  
// Generating and printing UUID_V4 2  
// 3  
id.uuid println 4
```

Read text file into a list and push result to a stack - *io.textfile*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Reading text file as a list of string, while taking filename from a stack and store result in the stack

```
1: function IO.TEXTFILE()
2:   ▷ Reading text file
3:   Filename ← current stack
4:   if Value = None then
5:     return Error("Stack is too shallow")
6:   current stack ← Call("Io.Textfile", [Filename])
```

```
// 1
// Print /etc/passwd on console 2
// 3
"/etc/passwd" io.textfile { println } loop 4
```

Read text file into a list and push result to a workbench - *io.textfile*.

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Reading text file as a list of string, while taking filename from workbench and store result to the workbench

```
1: function IO.TEXTFILE()
2:   ▷ Reading text file
3:   Filename ← workbench
4:   if Value = None then
5:     return Error("Stack is too shallow")
6:   workbench ← Call("Io.Textfile", [Filename])
```

```
// 1
// Print /etc/passwd on console 2
// 3
"/etc/passwd" . io.textfile. take { println } loop 4
```

Loading VM state from the WORLD file - *load*

Danger

Content from the external file will alter current VM state

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

With the help of the *save* function, users can store the current state of the BUND VM into an external file. This file, called the “WORLD” file, contains a frozen version of the registered lambda functions, user functions, aliases, and the content of the stacks. The *load* function, which can be used as `"WORLD_file" load`, will load content from this file and add it to the existing BUND state.

```
1: function BUND-LOAD()
2:   ▷ Restore VM state from WORLD file
3:   Filename ← current stack
4:   if Value = None then
5:     return Error("Stack is too shallow")
6:   BUND_LOAD(Filename)
```

```
// 1
// Load VM state from file _state.world_ 2
// 3
"state" load 4
```

Loading function aliases from the WORLD file - *load.aliases*



Danger

Content from the external file will alter current list of aliases

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

With the help of the *save* function, users can store the current state of the BUND VM into an external file. This file, called the “WORLD” file, contains a frozen version of the registered lambda functions, user functions, aliases, and the content of the stacks. The *load.aliases* function, which can be used as `"WORLD_file" load.aliases`, will load VM aliases from this file and add it to the list of existing aliases.

```
1: function BUND-LOAD-ALIASES()
2:   ▷ Restore VM function aliases from WORLD file
3:   Filename ← current stack
4:   if Value = None then
5:     return Error("Stack is too shallow")
6:   BUND_LOAD_ALIASES(Filename)
```

```
// Load aliases from file _state.world_
"state" load.aliases
```

1
2

Loading lambda functions from the WORLD file - *load.lambdas*

Danger

Content from the external file will alter current VM state

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

With the help of the *save* function, users can store the current state of the BUND VM into an external file. This file, called the “WORLD” file, contains a frozen version of the registered lambda functions, user functions, aliases, and the content of the stacks. The *load.lambdas* function, which can be used as `"WORLD_file" load.lambdas`, will load lambda functions from this file and add it to the list of existing lambdas.

```
1: function BUND-LOAD-LAMBDAS()
2:   ▷ Restore lambda functions from WORLD file
3:   Filename ← current stack
4:   if Value = None then
5:     return Error("Stack is too shallow")
6:   BUND_LOAD_LAMBDAS(Filename)
```

```
// Load lambda functions from file _state.world_
"state" load.lambdas
```

1
2

Loading for the stacks from the WORLD file - *load.stacks*

Danger

Content from the external file will alter current VM state

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

With the help of the *save* function, users can store the current state of the BUND VM into an external file. This file, called the “WORLD” file, contains a frozen version of the registered lambda functions, user functions, aliases, and the content of the stacks. The *load.stacks* function, which can be used as `"WORLD_file" load.stacks`, will load stacks content from this file and add it to the existing data contexts.

```
1: function BUND-LOAD-STACKS()  
2:   ▷ Restore VM state from WORLD file  
3:   Filename ← current stack  
4:   if Value = None then  
5:     return Error("Stack is too shallow")  
6:   BUND_LOAD_STACKS(Filename)
```

```
// Load stacks data context from file _state.world_  
"state" load.stacks
```

1
2

E to the power of the value taken from stack - *math.exp*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Raise mathematical constant e to the power of the value that is taken from stack

```
1: function MATH.EXP()  
2:   ▷ E to the power of X  
3:   X ← current stack  
4:   if X = None then  
5:     return Error("Stack is too shallow")  
6:   current stack ← Call("Math_Exp", [X])
```

```
//  
// e^1 is 2.718281828459045  
//  
1.0 math.exp println
```

1
2
3
4

Calculate factorial of the value taken from stack - *math.factorial*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Compute factorial of the numeric value taken from stack

```
1: function MATH.FACTORIAL()
2:   ▷ factorial of X
3:   X ← current stack
4:   if X = None then
5:     return Error("Stack is too shallow")
6:   current stack ← Call("Math_Factorial", [X])
```

```
//
// 1! is 1
//
1 math.factorial println
```

1
2
3
4

Calculate natural logarithm of the value taken from stack - *math.ln*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Compute natural logarithm of value stored in stack

```
1: function MATH.LN()
2:   ▷ Natural logarithm of X
3:   X ← current stack
4:   if X = None then
5:     return Error("Stack is too shallow")
6:   current stack ← Call("Math_Ln", [X])
```

```
//
// ln(1.0) is 0.0
//
1.0 math.ln println
```

1
2
3
4

Calculate n-th root of the value taken from stack - *math.nroot*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Compute N-th root of X, where N and X are taken from stack. Result is stored to the stack.

```
1: function MATH.NROOT()
2:   ▷ n-th root of X
3:   N ← current stack
4:   if N = None then
5:     return Error("Stack is too shallow")
6:   X ← current stack
7:   if X = None then
8:     return Error("Stack is too shallow")
9:   current stack ← Call("Math_NRoot", [X, N])
```

```
//
// nroot(1.0, 1.0) is 1.0
//
1.0 1.0 math.nroot println
```

1
2
3
4

Calculate perimeter of rectangle where X and Y are taken from stack - *math.perimeter*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

The `math.perimeter` function calculates the total length of a rectangle's boundaries, given its width and height, returning the result as a floating-point number.

```
1: function MATH.PERIMETER()
2:   ▷ Compute length of rectangle boundaries
3:   X ← current stack
4:   if N = None then
5:     return Error("Stack is too shallow")
6:   Y ← current stack
7:   if X = None then
8:     return Error("Stack is too shallow")
9:   current stack ← Call("Math_Perimeter", [X, Y])
```

```
// 1
// perimeter of X=1.0 and Y=1.0 is 4.0 2
// 3
1.0 1.0 math.perimeter println 4
```

Calculate Y power of X where X and Y are taken from stack - *math.power*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Computer power Y of X. Both values are taken from the stack

```
1: function MATH.POWER()
2:     ▷ n-th root of X
3:     N ← current stack
4:     if N = None then
5:         return Error("Stack is too shallow")
6:     X ← current stack
7:     if X = None then
8:         return Error("Stack is too shallow")
9:     current stack ← Call("Math_Power", [X, N])
```

```
// 1
// 1.0 in the power of 0.0 is 1.0 2
// 3
0.0 1.0 math.perimeter println 4
```

Perform Simple Moving Average smoothing for list of numbers - *math.smoothing*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Performing SMA (simple moving average) smoothing of numeric sample that is loaded from the stack. Result value is returned to the stack.

```
1: function MATH.SMOOTHING()
2:   ▷ SMA smoothing
3:   Sample ← current stack
4:   if Sample = None then
5:     return Error("Stack is too shallow")
6:   current stack ← Call("Math_Smoothing", [Sample])
```

```
// 1
// SMA smoothing 2
// 3
[1.22 3.004 5.0 4 2 88 3] math.smoothing println 4
```

Perform Simple Moving Average smoothing for list of numbers. Source is workbench - *math.smoothing*.

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Performing SMA (simple moving average) smoothing of numeric sample that is loaded from the workbench. Result value is returned to the workbench.

```
1: function MATH.SMOOTHING()  
2:   ▷ SMA smoothing  
3:   Sample ← workbench  
4:   if Sample = None then  
5:     return Error("Workbench is too shallow")  
6:   workbench ← Call("Math_Smoothing", [Sample])
```

```
// 1  
// SMA smoothing 2  
// 3  
[1.22 3.004 5.0 4 2 88 3] . math.smoothing. take println 4
```

Perform Simple Moving Average smoothing for list of numbers preserving original value in the workbench. Source is workbench - *math.smoothing._comma*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Performing SMA (simple moving average) smoothing of numeric sample that is loaded from the workbench. Original value is preserved in the workbench. Result value is returned to the workbench.

```
1: function MATH.SMOOTHING.,()
2:   ▷ SMA smoothing
3:   Sample ← workbench
4:   if Sample = None then
5:     return Error("Stack is too shallow")
6:   workbench ← Sample
7:   workbench ← Call("Math_Smoothing", [Sample])
```

```
// 1
// SMA smoothing 2
// 3
[1 2 3] . math.smoothing., debug.display_workbench 4
// Prints result and original value stored in the 5
workbench
```

Perform Simple Moving Average smoothing for list of numbers preserving original value in the stack - *math.smoothing_comma*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Performing SMA (simple moving average) smoothing of numeric sample that is loaded from the stack. Original value is preserved in the stack. Result value is returned to the stack.

```
1: function MATH.SMOOTHING,()
2:   ▷ SMA smoothing
3:   Sample ← current stack
4:   if Sample = None then
5:     return Error("Stack is too shallow")
6:   current stack ← Sample
7:   current stack ← Call("Math_Smoothing", [Sample])
```

```
// 1
// SMA smoothing 2
// 3
[1 2 3] math.smoothing, debug.display_stack 4
// Prints result and original value stored in the stack 5
```

Moving value from current stack to named stack - *move*

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Moving value from current stack to named stack

```
1: function FUNCTION-NAME()
2:   ▷ Move value from current stack
3:   Name ← current stack
4:   if Name = None then
5:     return Error("Stack is too shallow")
6:   Value ← current stack
7:   if Value = None then
8:     return Error("Stack is too shallow")
9:   Name stack ← Value
```

```
// 1
// 2
// 3
@A 4
@main 5
  42 :A move 6
@A 7
  42 == { "Data moving from @main to @A happens" 8
        println } if 9
```

Moving value between named stacks - *move_from*

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Moving value between two named stacks

```
1: function FUNCTION-NAME()
2:   ▷ Move value between named stacks
3:   Name_From ← current stack
4:   if Name_From = None then
5:     return Error("Stack is too shallow")
6:   Name_To ← current stack
7:   if Name_To = None then
8:     return Error("Stack is too shallow")
9:   Value ← Name_From stack
10:  if Value = None then
11:    return Error("Stack is too shallow")
12:  Name_To stack ← Value
```

```
// 1
// 2
// 3
@A 4
  42 5
@B 6
@main 7
  42 :B :A move_from 8
@B 9
  42 == { 10
    "Data moving from @A to @B happens" 11
    println } if 12
```

Rotate current stack to the left - *rotate_current_left*

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Rotate current stack as a circular buffer to the left

```
1: function ROTATE_CURRENT_LEFT()  
2:   ▷ Rotate current stack  
3:   Name ← VM::current_stack_name()  
4:   ROTATE_STACK_LEFT(1, Name)
```

```
// 1  
// Rotate current stack to the left 2  
// 3  
1 2 3 rotate_current_left 4  
// The state of stack is 2 3 1 5
```

Rotate current stack to the right - *rotate_current_right*

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Rotate current stack as a circular buffer to the right

```
1: function ROTATE_CURRENT_RIGHT()
2:     ▷ Rotate current stack
3:     Name ← VM::current_stack_name()
4:     ROTATE_STACK_RIGHT(1, Name)
```

```
// 1
// Rotate current stack to the right 2
// 3
1 2 3 rotate_current_right 4
// The state of stack is 3 1 2 5
```

Rotate named stack left - *rotate_stack_left*

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Rotate named stack as a circular buffer to the left

```
1: function ROTATE_CURRENT_LEFT()
2:     ▷ Rotate named stack
3:     Name ← current stack
4:     ROTATE_STACK_LEFT(1, Name)
```

```
// 1
// Rotate stack :A to the right and do the math 2
// 3
@main 4
@A 5
  1 2 41 6
@main 7
:A rotate_stack_left + println 8
// Printing result of + operation = 42 9
```

Rotate named stack right - *rotate_stack_right*

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Rotate named stack as a circular buffer to the right

```
1: function ROTATE_CURRENT_LEFT()
2:   ▷ Rotate named stack
3:   Name ← current stack
4:   ROTATE_STACK_RIGHT(1, Name)
```

```
// 1
// Rotate stack :A to the right and do the math 2
// 3
@main 4
@A 5
  1 41 3 6
@main 7
:A rotate_stack_right + println 8
// Printing result of + operation = 42 9
```

Saving state of the VM into a WORLD file - *save*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

With the help of the *save* function, users can store the current state of the BUND VM into an external file. This file, called the “WORLD” file, contains a frozen version of the registered lambda functions, user functions, aliases, and the content of the stacks.

```
1: function BUND-SAVE()
2:   ▷ Restore VM state from WORLD file
3:   Filename ← current stack
4:   if Value = None then
5:     return Error(“Stack is too shallow”)
6:   BUND_SAVE(Filename)
```

```
// 1
// Save VM state to the WORLD file _state.world_ 2
// 3
"state" save 4
```

Saving function aliases into a WORLD file - *save.aliases*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

With the help of the *save* function, users can store the current state of the BUND VM into an external file. This file, called the “WORLD” file, contains a frozen version of the registered lambda functions, user functions, aliases, and the content of the stacks. Function *save.aliases* performs the storing of current aliases into the WORLD file.

```
1: function BUND-SAVE-ALIASES()
2:   ▷ Restore VM state from WORLD file
3:   Filename ← current stack
4:   if Value = None then
5:     return Error(“Stack is too shallow”)
6:   BUND_SAVE_ALIASES(Filename)
```

```
// 1
// Save BUND function aliases to the WORLD file 2
_state.world_ 3
// 4
"state" save.aliases
```

Saving lambda functions into a WORLD file - *save.lambdas*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

With the help of the *save* function, users can store the current state of the BUND VM into an external file. This file, called the “WORLD” file, contains a frozen version of the registered lambda functions, user functions, aliases, and the content of the stacks.

```
1: function BUND-SAVE()
2:   ▷ Restore VM state from WORLD file
3:   Filename ← current stack
4:   if Value = None then
5:     return Error(“Stack is too shallow”)
6:   BUND_SAVE(Filename)
```

```
// 1
// Save lambdas to the WORLD file _state.world_ 2
// 3
"state" save.lambdas 4
```

Store script from the stack into bootstrap scripts repository of the WORLD file - *save.script*

Danger

This function will change the state of the bootstrap scripts which may change the state of the VM

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Adding script stored in the stack to the bootstrap scripts stored in the WORLD file.

```
1: function BUND-SAVE-SCRIPT()
2:   ▷ Saving bootstrap script into the WORLD file
3:   Filename ← current stack
4:   if Filename = None then
5:     return Error("Stack is too shallow")
6:   Snippet ← current stack
7:   if Snippet = None then
8:     return Error("Stack is too shallow")
9:   Name ← current stack
10:  if Name = None then
11:    return Error("Stack is too shallow")
12:  BUND_SAVE_SCRIPT(Filename, Snippet, Name)
```

BUND standard Library reference

```
// 1
// Adding Plus42 script to bootstrap scripts 2
// 3
:Plus42 "42 +" "sample" save.script 4
```

Saving stacks data into a WORLD file - *save.stacks*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

With the help of the *save* function, users can store the current state of the BUND VM into an external file. This file, called the “WORLD” file, contains a frozen version of the registered lambda functions, user functions, aliases, and the content of the stacks. Function *save.stacks* performs the save stacks data into the WORLD file.

```
1: function BUND-SAVE-STACKS()
2:   ▷ Restore VM state from WORLD file
3:   Filename ← current stack
4:   if Value = None then
5:     return Error(“Stack is too shallow”)
6:   BUND_SAVE_STACKS(Filename)
```

```
// 1
// Save stacks data into the WORLD file _state.world_ 2
// 3
"state" save.stacks 4
```

Generate a list of the sequence of floating point numbers, according to sample configuration stored in the stack - *seq*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Create list filled with sequence of numbers according to configuration stored in the stack. Currently supported types of the sequence is:

- ☐ seq.ascending and this is the default. Parameters are: X, Step, N
- ☐ seq.descending. Parameters are: X, Step, N

```
1: function SEQ()
2:   ▷ Creating sequence of float numbers
3:   Conf ← current stack
4:   if Conf = None then
5:     return Error("Stack is too shallow")
6:   current stack ← Call("Seq", [Conf])
```

```
// 1
// Generate descending sequence 2
// 3
config 4
  "type"  "seq.descending" set 5
  "X"     100.0          set 6
  "Step"  3.0            set 7
  "N"     3              set 8
"Generated sequence is: " print seq println 9
```

Generate a list of the sequence of floating point numbers sorted in ascending - *seq.asc*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Create list filled with sequence of numbers started from X with defined step and size. Ordered in ascending. Result is stored in stack.

```
1: function SEQ.ASC()
2:   ▷ Creating sequence of float numbers
3:   X ← current stack
4:   if X = None then
5:     return Error("Stack is too shallow")
6:   Step ← current stack
7:   if Step = None then
8:     return Error("Stack is too shallow")
9:   N ← current stack
10:  if N = None then
11:    return Error("Stack is too shallow")
12:  current stack ← Call("Seq_Asc", [X, Step, N])
```

```
//
// Create list containing [ 1.0 :: 1.1 :: 1.2 :: ]
//
3 0.1 1.0 seq.asc
```

1
2
3
4

Rotate circular list of stacks to left - *stacks_left*

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Rotate circular buffer of stacks to the left

```
1: function STACKS_LEFT()  
2:     ▷ Rotate list of stacks  
3:     STACKS_LEFT(1)
```

```
// 1  
// Rotate list of stacks 2  
// 3  
@A 4  
@B 5  
@C 6  
    stacks_left 7  
// Now stacks are in order B C A 8
```

Rotate circular list of stacks to right - *stacks_right*

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Rotate circular buffer of stacks to the right

```
1: function STACKS_RIGHT()
2:     ▷ Rotate list of stacks
3:     STACKS_RIGHT(1)
```

```
// 1
// Rotate list of stacks 2
// 3
@A 4
@B 5
@C 6
    stacks_right 7
// Now stacks are in order C A B 8
```

Check if stack exists - *stack_exists*

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Returning TRUE to the stack if named stack exists, FALSE - otherwise

```
1: function STACK_EXISTS()  
2:     ▷ Check if stack exists  
3:     Name ← current stack  
4:     if Name = None then  
5:         return Error("Stack is too shallow")  
6:     if Not Call(Stack_Exists, [Name]) then  
7:         current stack ← FALSE  
8:     else  
9:         current stack ← TRUE
```

```
// 1  
// This snippet will check if stack with name "A" 2  
// exists and prints the message 3  
// 4  
@A 5  
:A 6  
    stack_exists 7  
    { "Stack A existing" } ? 8  
// And yes, it ddoes exists, as @A will make sure that 9  
// it does  
:B 10  
    stack_exists 11  
    not { "There is no stack with name B" } ? 12  
// And stack B doesn't exists 13
```

Convert string stored on stack to the list of tokens - *string.tokenize*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Convert string stored in stack to the list of lexical tokens. Result is stored on stack.

```
1: function STRING.TOKENIZE.STACK()
2:   ▷ String tokenization
3:   Text ← current stack
4:   if Text = None then
5:     return Error("Stack is too shallow")
6:   current stack ← Call("String.Tokenize", [Text])
```

```
// 1
// Print tokens from the string 2
// 3
"Hello world, and Привет мир!" string.tokenize println 4
```

Convert string stored in workbench to the list of tokens - *string.tokenize*.

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Convert string stored on workbench to the list of lexical tokens. Result is stored into workbench.

```
1: function STRING.TOKENIZE.WB()
2:   ▷ String tokenization
3:   Text ← workbench
4:   if Text = None then
5:     return Error("Workbench is too shallow")
6:   workbench ← Call("String.Tokenize", [Text])
```

```
// 1
// Print tokens from the string 2
// 3
"Hello world, and Привет мир!" . string.tokenize. { 4
  println 5
} loop 6
```

Convert string stored on stack to the stemmed list of tokens - *string.tokenize.stemmed*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Convert string stored in stack to the list of stemmed lexical tokens. Result is stored on stack.

```
1: function STRING.TOKENIZE.STEMMED.STACK()
2:   ▷ String tokenization
3:   Text ← current stack
4:   if Text = None then
5:     return Error("Stack is too shallow")
6:   current stack ← Call("String.Tokenize.Stemmed", [Text])
```

```
// 1
// Print tokens from the string 2
// 3
"Hello world, and peace to all!" string.tokenize.stemmed 4
println
```

Convert string stored in workbench to the list of stemmed tokens - *string.tokenize.stemmed*.

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Convert string stored on workbench to the list of unique and stemmed lexical tokens. English language is a primary language for stemming. Result is stored into workbench.

```
1: function STRING.TOKENIZE.STEMMED.WB()
2:   ▷ String tokenization
3:   Text ← workbench
4:   if Text = None then
5:     return Error("Workbench is too shallow")
6:   workbench ← Call("String.Tokenize.Stemmed", [Text])
```

```
// 1
// Print tokens from the string 2
// 3
"Peace on Earth, good will to man !" . 4
string.tokenize.stemmed. {
  println 5
} loop 6
```

Convert string stored on stack to the unique list of tokens - *string.tokenize.unique*

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Convert string stored in stack to the list of unique lexical tokens. Result is stored on stack.

```
1: function STRING.TOKENIZE.UNIQUE.STACK()
2:   ▷ String tokenization
3:   Text ← current stack
4:   if Text = None then
5:     return Error("Stack is too shallow")
6:   current stack ← Call("String.Tokenize.Unique", [Text])
```

```
// 1
// Print tokens from the string 2
// 3
"Hello world, and Привет мир!" string.tokenize.unique 4
println
```

Convert string stored in workbench to the list of unique tokens - *string.tokenize.unique*.

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Convert string stored on workbench to the list of unique lexical tokens. Result is stored into workbench.

```
1: function STRING.TOKENIZE.UNIQUE.WB()
2:   ▷ String tokenization
3:   Text ← workbench
4:   if Text = None then
5:     return Error("Workbench is too shallow")
6:   workbench ← Call("String.Tokenize.Unique", [Text])
```

```
// 1
// Print tokens from the string 2
// 3
"Hello world, and Привет мир!" . string.tokenize.unique. 4
{
  println 5
} loop 6
```

Swap two vallues in current stack - *swap_one*

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Swapping two values in the current stack

```
1: function SWAP_ONE()
2:     ▷ Swapping data in stack
3:     Value1 ← current stack
4:     if Value1 = None then
5:         return Error("Stack is too shallow")
6:     Value2 ← current stack
7:     if Value2 = None then
8:         return Error("Stack is too shallow")
9:     current stack ← Value2
10:    current stack ← Value1
```

```
// 1
// Swapping values 2
// 3
1 2 swap_one 4
// As the result, we will have following state in the 5
stack 2 1
```

Execute command in system shell and push result as string to stack - *system.shell*

Danger

External application will be executed in default system's shell and with current user privileges. Running this command as "root" is a bad idea.

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Execute external command in system's default shell and store result as string on the stack

Aliases

- ☐ sh

```
1: function SYSTEM.SHELL()
2:     ▷ Execute an external command
3:     Cmd ← current stack
4:     if Cmd = None then
5:         return Error("Stack is too shallow")
6:     current stack ← Call("System.Shell", [Cmd])
```

BUND standard Library reference

```
// 1
// Finding out our uname 2
// 3
"uname -a" sh println 4
```

Execute command in system shell and push result as string to workbench - *system.shell*.



Danger

External application will be executed in default system's shell and with current user privileges. Running this command as "root" is a bad idea.

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☒ bund runtime

Execute external command in system's default shell and store result as string to workbench

Aliases

- ☐ sh.

```
1: function SYSTEM.SHELL.WB()
2:     ▷ Execute an external command
3:     Cmd ← workbench
4:     if Cmd = None then
5:         return Error("Stack is too shallow")
6:     workbench ← Call("System.Shell", [Cmd])
```

BUND standard Library reference

```
// 1
// Finding out our uname 2
// 3
"uname -a" . sh. take println 4
```

Make existing stack with name - current - *to_current*

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Taking name of the stack from the stack and makes this stack a current stack. If stack doesn't exists raise an error

```
1: function TO_CURRENT()
2:   ▷ Make already existing stack a current stack
3:   Value ← current stack
4:   if Value = None then
5:     return Error("Stack is too shallow")
6:   TO_CURRENT(Name)
```

```
// 1
// Make stack with name "A" - current 2
// stack must already exists          3
// 4
"A" to_current                        5
```

Make stack with name - *current* - *to_stack*

Defined in

- ☒ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Taking name of the stack from the stack and makes this stack a current stack. If stack doesn't exist VM creates it.

```
1: function TO_CURRENT()
2:   ▷ Make stack a current stack. Create if not existing.
3:   Name ← current stack
4:   if Name = None then
5:     return Error("Stack is too shallow")
6:   if not VM::stack_exists(Name) then
7:     ENSURE_STACK(Name)
8:   TO_CURRENT(Name)
```

```
//
// Make stack with name "A" - current
//
"A" to_stack
```

1
2
3
4

Load and execute bund scripts from external file. - *use*

Danger

Function *use* may change the state of the VM as it executes functions from an external file.

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Function *use* loads and executes BUND script. The file name is passed through the stack.

```
1: function BUND-USE-FROM-STACK()
2:   ▷ Load and execute external scripts
3:   Filename ← current stack
4:   if Value = None then
5:     return Error("Stack is too shallow")
6:   BUND_USE(Filename)
```

```
// 1
// Load and execute script from "sample.bund" 2
// 3
"sample.bund" use 4
```

Load and execute bund scripts from external file. - *use*.

Danger

Function *use* may change the state of the VM as it executes functions from an external file.

Defined in

- ☐ rust_multistack
- ☐ rust_multistackvm
- ☐ bund runtime

Function *use*. loads and executes BUND script. The file name is passed through the workbench.

```
1: function BUND-USE-FROM-WORKBENCH()
2:   ▷ Load and execute external scripts
3:   Filename ← workbench
4:   if Value = None then
5:     return Error("Stack is too shallow")
6:   BUND_USE(Filename)
```

```
// 1
// Load and execute script from "sample.bund" 2
// 3
"sample" ".bund" + . use. 4
```

Conclusion

BUND is a very new language. It is currently in its early stages of development, and the language's runtime has many limitations. The standard library requires improvement, and the author or contributor must address several potential bugs. However, the *bundcore* crate and its dependencies have successfully passed all their test cases, which is a promising sign. Although the language is simple and its underlying dependencies are generally stable, there are no guarantees against critical bugs. The license is attached for reference. While concatenative, stack-based programming languages are not widely used in general programming practices, they have stood the test of time and deserve more attention from the software development community. BUND aims to address design gaps in this concept, and the author hopes to spark interest with his ideas and inspirations that brought BUND into existence.

You can get in touch with my via [in](#) my LinkedIn profile.
The BUND project is hosted on my GitHub page [vulogov](#)



License

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.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets “[]” replaced with your own identifying information. (Don’t include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same “printed page” as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

BUND standard Library reference

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Content

Introduction	3
What exactly is a Bund?	5
Circular data stack	6
Stack-of-stacks references	7
Workbench	8
What does the word “bund” mean?	9
How to use the reference?	11
BUND Standard library reference	13
Execute code snippet in the BUND debugger - <i>debug</i>	15
Arguments passed via CLI for script and shell subcommands - <i>args</i> .	
16	
Parsing arguments passed via CLI for script and shell subcommands	
- <i>args.parse</i>	17
Load and execute scripts stored as bootstrap scripts in the WORLD	
file - <i>bootstrap</i>	18
Evaluate BUND code snippet taken as string from stack - <i>bund.eval</i>	
19	
Evaluate BUND code snippet taken from the file. The name of the	
file is taken as string from stack - <i>bund.eval-file</i>	20
Evaluate BUND code snippet taken from the file. The name of the	
file is taken as string from workbench - <i>bund.eval-file</i>	21
Evaluate BUND code snippet taken as string from workbench -	
<i>bund.eval</i>	22
Terminate BUND interpreter and exit - <i>bund.exit</i>	23
Clear current stack - <i>clear</i>	24
Clear named stack - <i>clear_in</i>	25
Taking value from stack and convert it to boolean - <i>convert.to_bool</i> .	
26	
Taking value from workbench and convert it to boolean -	

<i>convert.to_bool.</i>	27
Taking value from stack and convert it to float - <i>convert.to_float</i>	28
Taking value from workbench and convert it to float - <i>convert.to_float.</i>	29
Taking value from stack and convert it to int - <i>convert.to_int</i>	30
Taking value from workbench and convert it to int - <i>convert.to_int.</i>	31
Taking value from stack and convert it to list - <i>convert.to_list</i>	32
Taking value from workbench and convert it to list - <i>convert.to_list.</i>	33
Taking value from stack and convert it to matrix - <i>convert.to_matrix</i>	34
Taking value from workbench and convert it to matrix - <i>convert.to_matrix.</i>	35
Taking value from stack and convert it to string - <i>convert.to_string.</i>	36
Taking value from workbench and convert it to string - <i>convert.to_string.</i>	37
Taking value from stack and convert it to textbuffer - <i>convert.to_textbuffer</i>	38
Taking value from workbench and convert it to textbuffer - <i>convert.to_textbuffer.</i>	39
Return name of current stack to current stack - <i>current</i>	40
Display information about host - <i>debug.display_hostinfo</i>	41
Display data stored in current stack - <i>debug.display_stack</i>	42
Display data stored in workbench - <i>debug.display_workbench</i>	43
Drop into a debug shell - <i>debug.shell</i>	44
Decoding string from stack from BASE64 - <i>decode.base64</i>	45
Decoding string from workbench from BASE64 - <i>decode.base64.</i>	46
Drop element from the stack - <i>drop</i>	47
Drop the last value in the named stack - <i>drop_in</i>	48
Remove stack with all data - <i>drop_stack</i>	49
Duplicate multiple values in the current stack - <i>dup_many</i>	50
Duplicate multiple values in the named stack - <i>dup_many_in</i>	51
Duplicate single value in the current stack - <i>dup_one</i>	52

Duplicate single value in the named stack - <i>dup_one_in</i>	53
Encoding string from stack into BASE64 - <i>encode.base64</i>	54
Encoding string from workbench into BASE64 - <i>encode.base64</i> .	55
Make named stack current, create if stack doesn't exists - <i>ensure_stack</i>	56
Make named stack with set capacity current, create if stack doesn't exists. - <i>ensure_stack_with_capacity</i>	57
Writing string value of the data loaded from the stack into file - <i>file.write</i>	58
Folding all values in the current stack - <i>fold</i>	59
Folding all values in the named stack - <i>fold_stack</i>	60
Returning path of current work directory - <i>fs.cwd</i>	61
Generate random v4 UUID - <i>id.uuid</i>	62
Read text file into a list and push result to a stack - <i>io.textfile</i>	63
Read text file into a list and push result to a workbench - <i>io.textfile</i> .	64
Loading VM state from the WORLD file - <i>load</i>	65
Loading function aliases from the WORLD file - <i>load.aliases</i>	66
Loading lambda functions from the WORLD file - <i>load.lambdas</i> .	67
Loading for the stacks from the WORLD file - <i>load.stacks</i>	68
E to the power of the value taken from stack - <i>math.exp</i>	69
Calculate factorial of the value taken from stack - <i>math.factorial</i>	70
Calculate natural logarithm of the value taken from stack - <i>math.ln</i>	71
Calculate n-th root of the value taken from stack - <i>math.nroot</i> ...	72
Calculate perimeter of rectangle where X and Y are taken from stack - <i>math.perimeter</i>	73
Calculate Y power of X where X and Y are taken from stack - <i>math.power</i>	74
Perform Simple Moving Average smoothing for list of numbers - <i>math.smoothing</i>	75
Perform Simple Moving Average smoothing for list of numbers. Source is workbench - <i>math.smoothing</i>	76
Perform Simple Moving Average smoothing for list of numbers preserving original value in the workbench. Source is workbench -	

<i>math.smoothing_comma</i>	77
Perform Simple Moving Average smoothing for list of numbers preserving original value in the stack - <i>math.smoothing_comma</i>	78
Moving value from current stack to named stack - <i>move</i>	79
Moving value between named stacks - <i>move_from</i>	80
Rotate current stack to the left - <i>rotate_current_left</i>	81
Rotate current stack to the right - <i>rotate_current_right</i>	82
Rotate named stack left - <i>rotate_stack_left</i>	83
Rotate named stack right - <i>rotate_stack_right</i>	84
Saving state of the VM into a WORLD file - <i>save</i>	85
Saving function aliases into a WORLD file - <i>save.aliases</i>	86
Saving lambda functions into a WORLD file - <i>save.lambdas</i>	87
Store script from the stack into bootstrap scripts repository of the WORLD file - <i>save.script</i>	88
Saving stacks data into a WORLD file - <i>save.stacks</i>	90
Generate a list of the sequence of floating point numbers, according to sample configuration stored in the stack - <i>seq</i>	91
Generate a list of the sequence of floating point numbers sorted in ascending - <i>seq.asc</i>	92
Rotate circular list of stacks to left - <i>stacks_left</i>	93
Rotate circular list of stacks to right - <i>stacks_right</i>	94
Check if stack exists - <i>stack_exists</i>	95
Convert string stored on stack to the list of tokens - <i>string.tokenize</i> .	96
Convert string stored in workbench to the list of tokens - <i>string.tokenize</i> .	97
Convert string stored on stack to the stemmed list of tokens - <i>string.tokenize.stemmed</i>	98
Convert string stored in workbench to the list of stemmed tokens - <i>string.tokenize.stemmed</i> .	99
Convert string stored on stack to the unique list of tokens - <i>string.tokenize.unique</i>	100
Convert string stored in workbench to the list of unique tokens - <i>string.tokenize.unique</i> .	101
Swap two vallues in current stack - <i>swap_one</i>	102

BUND standard Library reference

Execute command in system shell and push result as string to stack - <i>system.shell</i>	103
Execute command in system shell and push result as string to workbench - <i>system.shell</i>	105
Make existing stack with name - current - <i>to_current</i>	107
Make stack with name - current - <i>to_stack</i>	108
Load and execute bund scripts from external file. - <i>use</i>	109
Load and execute bund scripts from external file. - <i>use.</i>	110
Conclusion	111
License	113

