Aseba Documentation

Release 1.6

Stéphane Magnenat and other contributors

Jul 03, 2018

Aseba:

CHAPTER 1

About Aseba

Aseba is a set of tools which allow novices to program robots easily and efficiently. For these reasons, Aseba is well-suited for robotic education and research. Aseba is an open-source software created by Dr. Stéphane Magnenat with contributions from the community.

More Specifically, Aseba is an event-based architecture for real-time distributed control of mobile robots.It targets integrated multi-processor robots or groups of single-processor units, real or simulated.

The core of Aseba is a lightweight virtual machine tiny enough to run even on microcontrollers. With Aseba, we program robots in a user-friendly programming language using a cozy integrated development environment.

1.1 Node

In Aseba, there can be several robots or a robot with multiple processors running in the same network. This network can be software (TCP), hardware (CAN), or a mix of both. Each processor within a network runs a small virtual machine and is called a *node*.

Each node has its own tab in Aseba Studio, allowing you to program them independently, but with possible interactions through events.

1.2 Event

Aseba is an event-based architecture, which means that events trigger code execution asynchronously. Events have an identifier and optional payload data.

Aseba nodes can exchange events, and these can be of two types.

- The events that Aseba nodes exchange within an Aseba network are called **global events**.
- The events that are internal to a node are called **local events**. An example of a local event is the one emitted by a sensor that provides updated data.

If the code for receiving an event is defined, then the corresponding block of code is executed when the event is received. Code can also emit events, which can trigger the execution of code on another node or enable communication with an external program.

To start the execution of related code upon receiving new events, programs must not block and thus must not contain any infinite loops.

For instance, in the context of robotics, where a traditional robot control program would do some processing inside an infinite loop, an Aseba program would just do the processing inside a sensor-related event.

CHAPTER 2

Aseba Studio

Aseba Studio is an integrated development environment within which we edit and debug the programs of all nodes of an Aseba network.

2.1 The environment

2.1.1 Concurrent editing

Within Studio, each node of an Aseba network has its own tab with its program, memory content, execution status, and debugging commands. In addition, a toolbar provides general commands which affect all the nodes. This allows both an overall control of the network and a specific control of each node.

2.1.2 Powerful editor

The program editor provides syntax highlighting, indentation of blocks, and dragging of variables' names from memory. It also shows the current position of execution in step by step mode and colours errors in red.

2.1.3 Instant compilation

Studio recompiles the program while the developer is typing it. The result of compilation (success or a description of the error) is displayed below the editor. This permits the correction of errors as soon as they appear, which increases the quality of the code.

2.1.4 Data inspection

Studio lists the variables available on each node with their values. We can update this list in a single click. This list provides a quick overview of the state of the node.

2.1.5 Debugger

Studio has an integrated debugger; for each node, it gives the current execution status. It supports continuous execution, step by step, and breakpoints. A right click inside the program editor allows to set or clear a breakpoint on a specific line. After a breakpoint or a step, the values of the variables are updated.

2.1.6 Constants definition

We can define constants which are available on all the nodes. Constants can be re-ordered by drag and drop.

2.1.7 Network-wide events

We can specify the names of the events, and by double-clicking on a name, we can send the corresponding event. A right-click on an event allows to plot this event over time. Events can be re-ordered by drag and drop.

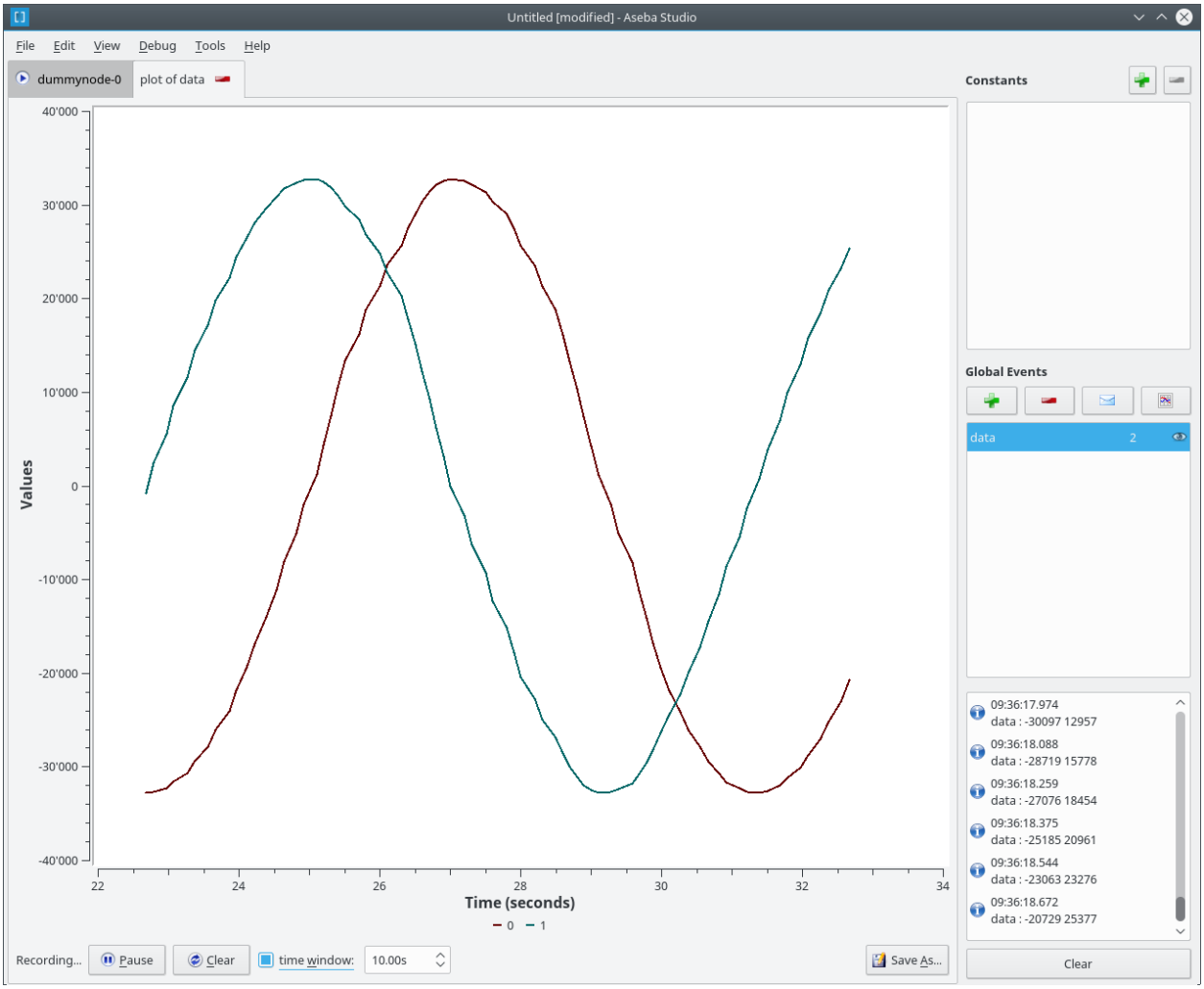
Below the list of event names, a log shows the recent events along with their time stamps and parameters. This allows the monitoring of the distributed behaviour in the network.

2.1.8 Local events, native functions and plugins

Studio shows the local events available on each node. It also lists its native functions. The tool-tip of each function gives a short documentation. A list of node-specific plugins is also provided, if any.

2.2 Event plotting in Aseba Studio

Aseba Studio allows the data of global events to be plotted over time. To do so, in a node tab, right click on a global event and select “Plot event data”. This creates a new tab with a 2D plot of the event data over time:



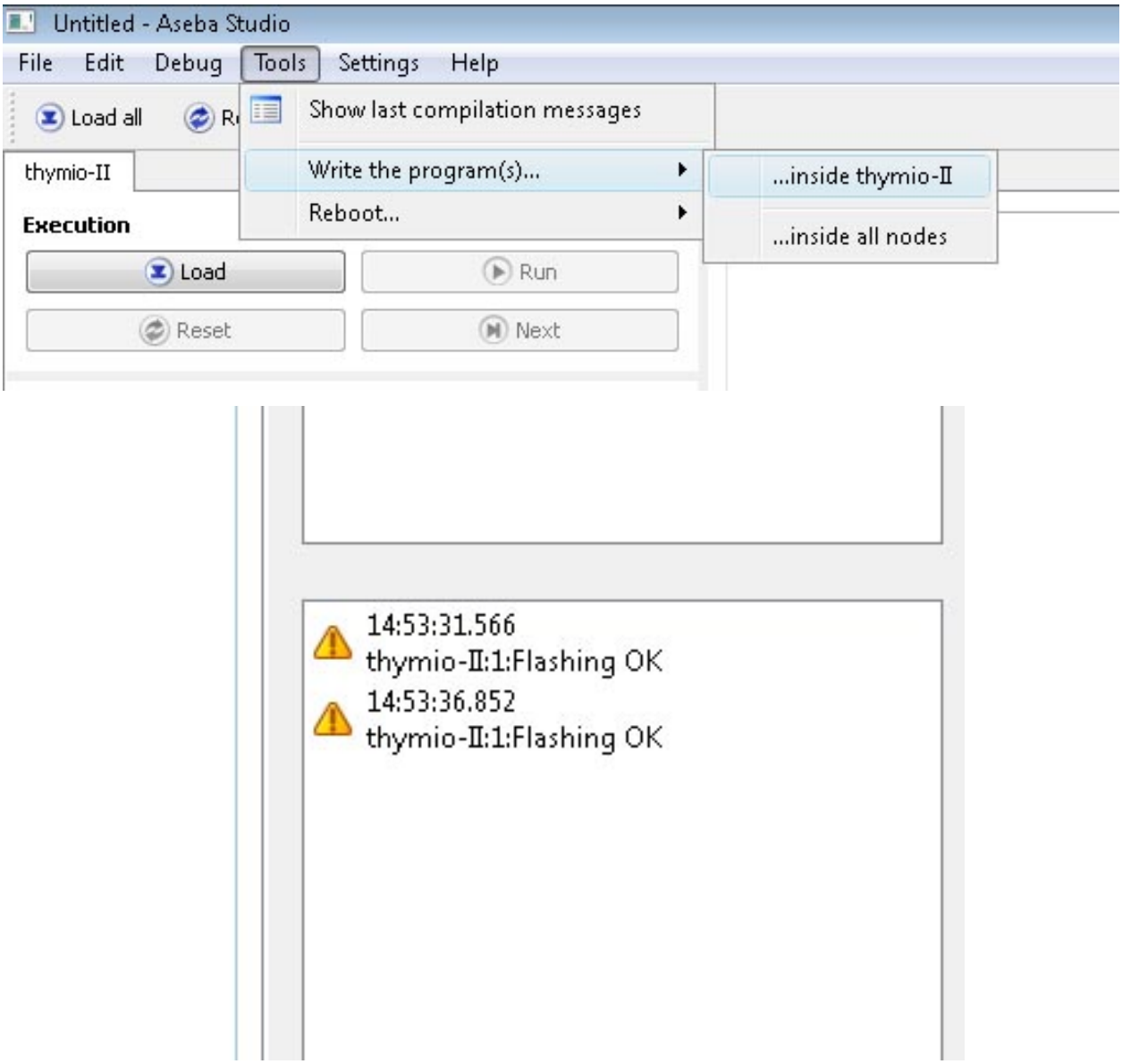
The y-axis scales automatically. The x-axis spans the whole duration of event data recording, with a maximum of *time window* if the latter is set. The recording can be started and stopped, and the data can be cleared. At the bottom-right part of the tab, a *Save As* button allows the event data to be saved in a text file. Each line will contain a timestamp and the values of the event data, separated by a space.

2.3 Flashing user code in Thymio

It is possible to save a user code in Thymio II so that, even when turned off, Thymio keeps the code. The saved behaviour will be accessible through the menu, the colour of your code is "colourless". Once this code is launched, the user cannot go back in the selection menu unless the robot is restarted.

2.3.1 To save the code in Thymio II

- Load the code, **without running it**.
- Click on "Tools" -> "Write the program(s) . . . " -> ". . . inside Thymio-II".
- Check in the window of the event if "Flashing OK" appears.



2.4 Tips

2.4.1 Save time

You can drag the names of variables, native functions, events, and constants and thus save typing.

2.4.2 Get help

The `Help` menu of Studio gives instant access to this documentation and to the documentation of the Aseba programming language.

CHAPTER 3

The Aseba language

The Aseba language syntax resembles that of a popular class of programming languages, including Pascal and Matlab (a common scientific programming language) for instance; we expect this similarity to allow developers with previous knowledge of any of these languages to feel quickly at ease with Aseba and thus lower the learning curve. Semantically, it is a simple imperative programming language with a single basic data type (16 bit signed integers) and arrays. This simplicity allows developers to program behaviours with no prior knowledge of a type system, integers being the most natural type of variables and well suited to programming microcontroller-based mobile robots.

3.1 Comments

Comments allow the adding of information which is ignored by the compiler. Comments are very useful to annotate the code with human-readable notes or to temporarily disable some code. Comments begin with a # and terminate at the end of the line.

Example:

```
# this is a comment
var b # another comment
```

Comments spanning several lines are also possible. They begin with #* and end with *#. Example:

```
#*
this is a comment spanning
several lines
*#
var b # a simple comment
```

3.2 Scalars

Scalar values are used in Aseba to represent numbers. They can be used in any expressions, like the initialisation of a variable, a mathematical expression or in a logical condition. The values are comprised between -32768 and 32767,

which is the range of 16 bit signed integers.

3.2.1 Notation

Scalars can be given using several radices. The most natural way is the decimal system, using digits from 0 to 9. Negative numbers are declared using a − (minus) sign preceding the number.

```
i = 42
i = 31415
i = -7
```

Both binary and hexadecimal numbers can also be used. Binary numbers are prefixed by `0b`, whereas hexadecimal numbers are prefixed by `0x`.

```
# binary notation
i = 0b110          # i = 6
i = 0b11111111     # i = 255

# hexadecimal notation
i = 0x10     # i = 16
i = 0xff     # i = 255
```

In binary notation, values are comprised between `0b0000000000000000` and `0b1111111111111111`, while in hexadecimal they are comprised between `0x0` and `0xffff`. Values that would be over 32767 in decimal are interpreted as negative numbers.

3.2.2 Variables

Variables refer either to single scalar values or to arrays of scalar values. You must declare all user-defined variables using the keyword `var` at the beginning of the Aseba programme before doing any processing.

The name of a variable is defined by these rules:

- The name can only contain upper or lower case alphanumeric characters, '_' or '.'
- The name must start with a valid alphabetic character or '_'
- The name is case sensitive: a variable named "foo" is different from one named "Foo"
- The name cannot be identical with one of Aseba's keywords (see *reserved keywords*)

Variables may be initialised in the declaration, using the assignment symbol and combined with any valid mathematical expression. A variable without any prior initialisation may have a random value, it should never be assumed to be zero.

Example:

```
var a
var b = 0
var c = 2*a + b         # warning: 'a' is not initialised
```

3.2.3 Reserved keywords

The following keywords cannot be used as valid names for variables, as they are already used by the Aseba language.

- *abs*
- *call*

- *callsub*
- *do*
- *else*
- *elseif*
- *emit*
- *end*
- *for*
- *if*
- *in*
- *onevent*
- *return*
- *step*
- *sub*
- *then*
- *var*
- *when*
- *while*

3.2.4 Constants

Constants can be defined in Aseba Studio using the "Constants" panel, but they cannot be defined directly in the code. A constant represents a numeric value which can be used wherever a number can be used. But unlike a variable, a constant cannot be modified during execution. Constants are useful when you want to easily change the behaviour between different executions, such as to adapt a threshold value, with a scope spanning several Aseba nodes. A constant cannot have the same name as a variable, otherwise an error is raised. By convention, a constant is often written in upper case.

```
# assuming a constant named THRESHOLD
var i = 600

if i > THRESHOLD then
    i = THRESHOLD - 1
end
```

3.2.5 Arrays

Arrays represent a contiguous area in memory, addressed as a single logical entity. The size of an array is fixed and must be specified in the declaration. Arrays can be declared using the usual square bracket operator `[]`. The number between the square brackets specifies the number of elements to be assigned to the array, thereafter referred to as its size. It can be any constant expression, including mathematical operations using scalars and constants. An optional assignment can be made using the array constructor (see below). If this is done, the size of the array need not be specified.

Example:

```
var a[10]               # array of 10 elements
var b[3] = [2,3,4]      # initialisation
var c[] = [3,1,4,1,5]   # implicit size of 5 elements
var d[3*FOO-1]          # size declared using a constant expression (FOO is a constant)
```

Arrays can be accessed in several ways:

- A single element is accessed by using the square bracket operator with a single value. Array indexes begin at zero. Any expression can be used as index, including mathematical expressions involving other variables.
- A range of elements can be accessed by using the square bracket operator with two constant expressions separated by a colon ':'. The validity of the range is checked at compile-time.
- If the square brackets are omitted, the entire array is accessed.

Example:

```
var foo[5] = [1,2,3,4,5]
var i = 1
var a
var b[3]
var c[5]
var d[5]

a = foo[0]         # copy first element from 'foo' to 'a'
a = foo[2*i-2]     # same
b = foo[1:3]       # take 2nd, 3rd and 4th elements of 'foo', copy to 'b'
b = foo[1:2*2-1]   # same
c = foo            # copy 5 elements from array 'foo' to array 'c'
d = c * foo        # multiply arrays 'foo' and 'c' element by element, copy result to
↪'d'
```

A scalar variable is considered to be an array of size one so the following code is legal:

```
var a[1] = [7]
var b = 0
b = a
```

3.3 Array constructors

Array constructors are a way to build arrays from variables, other arrays, scalars, or even complex expressions. They are useful in several cases, for example when initialising another array, or as operands in expressions, functions and events. An array constructor is made by using square brackets enclosing several expressions separated by a `,` (comma). The size of an array constructor is the sum of the sizes of the individual elements, and it must match the size of the array in which the result is stored.

Example:

```
var a[5] = [1,2,3,4,5]  # array constructor to initialise an array
var b[3] = [a[1:2],0]   # results in array b initialised to [2,3,0]
a = a + [1,1,1,1,1]     # add 1 to each element of array a
a = [b[1]+2,a[0:3]]     # results in [5,2,3,4,5]
```

3.4 Expressions and assignments

Expressions allow mathematical computations and are written using the normal mathematical infix syntax. Assignments use the keyword = and set the result of the computation of an expression into a scalar variable, an array element or a whole array, depending on the size of the operands. Aseba provides several operators. Please refer to the table below for a brief description, as well as for the precedence of each operator. To evaluate an expression in a different order, pairs of parentheses can be used to group sub-expressions.

Precedence	Operator	Description	Associativity	Arity
1	()	Group a sub-expression		unary
	[]	Index an array		unary
	•	Unary minus		unary
	~	Binary not		unary
	abs	Absolute value		unary
2	* /	Multiplication, division		binary
	%	Modulo		binary
3	• –	Addition, subtraction		binary
4	<< >>	Left shift, right shift		binary
5	&	Binary and	Left associative	binary
6	^	Binary exclusive or (xor)	Left associative	binary
7		Binary or	Left associative	binary
8	== != < <= > >=	Condition		binary
9	not	Logical not †		unary
10	and	Logical and †		binary
11	or	Logical or †		binary
12	=	Assignment		binary
	^= &=	Assignment by binary or, xor, and		binary
	*= /=	Assignment by product and quotient		binary
	%=	Assignment by modulo		binary
	+= -=	Assignment by sum and difference		binary
	<<= >>=	Assignment by left / right shift		binary
	++ –	Unary increment and decrement		unary

Footnotes † Only available from within `if`, `when`, and `while` structures ‡ Only available as statements, such as `a--` or `a[i]++`, not within an expression

The *assignment by* versions of the binary operators work by applying the operator to a variable and storing the result in this same variable. For instance, `A *= 2` is equal to `A = A * 2`. These short-cuts aim at making the code more readable.

Example:

```
a = 1 + 1
# Result: a = 2
a *= 3
# Result: a = 6
a++
# Result: a = 7

b = b + d[0]
b = (a - 7) % 5
c[a] = d[a]
c[0:1] = d[2:3] * [3,2]
```

3.4.1 Usage

Mathematical expressions are a general tool. As such, they can be used in a great variety of situations. Just to mention a few:

- On the right side of an assignment
- As an index when accessing elements of arrays
- Inside function calls
- As argument when emitting an event

3.5 Flow control

3.5.1 Conditionals

Aseba provides two types of conditionals statements: `if`-statements and `when`-statements. A conditional statement consists of a conditional expression and blocks of code. Conditional expressions are formed from a comparison operator and two operands which are arithmetic expressions; for example, `a < b+3` is a conditional expression. The following table lists the comparison operators:

Operator	Truth value
==	true if operands are equal
!=	true if operands are different
>	true if first operand is strictly larger than the second one
>=	true if the operand is larger or equal to the second one
<	true if first operand is strictly smaller than the second one
<=	true if the operand is smaller or equal to the second one

A conditional expression may also be formed by combining comparison expressions with the logical operators `and` (logical conjunction), `or` (logical disjunction) and `not` (logical negation); for example, `(a < b+3) or (a < 0)`. Precedence can be controlled by parentheses; for example `((a < b) or (b < c)) and ((d < e) or (e < f))`. While the Aseba language does not have boolean variables or literals — so you cannot write `flag = true` or `if flag then` — the result of a comparison is considered to be a boolean value (true or false) that can be used with the logical operators. Conditional expressions are also used in `while`-statements (see section *loops*).

Both `if` and `when` execute a different block of code according to whether a condition is true or false; but `when` executes the block corresponding to true only if the previous evaluation of the condition was false and the current one is true. This allows the execution of code only when something changes. The `if` conditional executes a first block of

code if the condition is true, a second block of code to execute if the condition is false can be added using the `else` keyword. Furthermore, additional conditions can be chained using the `elseif` keyword.

Example:

```
if a - b > c[0] then
    c[0] = a
elseif a > 0 then
    b = a
else
    b = 0
end

if a < 2 and a > 2 then
    b = 1
else
    b = 0
end

when a > b do
    leds[0] = 1
end
```

Here the `when` block executes only when `a` *becomes* larger than `b`. ### Loops

Two constructs allow the creation of loops: `while` and `for`.

A `while` loop repeatedly executes a block of code as long as the condition is true. The condition is of the same form as the one `if` uses.

Example:

```
while i < 10 do
    v = v + i * i
    i = i + 1
end
```

A `for` loop allows a variable to iterate over a range of integers, with an optional step size.

Example:

```
for i in 1:10 do
    v = v + i * i
end
for i in 30:1 step -3 do
    v = v - i * i
end
```

The value of the loop variable is undefined after the execution of the loop. It will usually be the last value + step, but can take another value due to optimisations, for instance in single-element loops.

3.6 Blocks

3.6.1 Subroutines

When you want to perform the same sequence of operations at two or more different places in the code, you can write common code just once in a subroutine and then call this subroutine from different places. You define a subroutine

using the `sub` keyword followed by the name of the subroutine. You call the subroutine using the `callsub` keyword, followed by the name of the subroutine. Subroutines cannot have arguments, nor be recursive, either directly or indirectly. Subroutines can access any variable.

Example:

```
var v = 0

sub toto
v = 1

onevent test
callsub toto
```

3.7 Events

Aseba is an event-based architecture, which means that events trigger code execution asynchronously. Events can be external, for instance a user-defined event coming from another Aseba node, or internal, for instance emitted by a sensor that provides updated data. The reception of an event executes, if defined, the block of code that begins with the `onevent` keyword followed by the name of the event; the code at the top of the programme is executed when the programme is started or reset.

To allow the execution of related code upon new events, programmes must not block and thus must not contain any infinite loop. For instance in the context of robotics, where a traditional robot control programme would do some processing inside an infinite loop, an Aseba programme would just do the processing inside a sensor-related event.

Example:

```
var run = 0

onevent start
run = 1

onevent stop
run = 0
```

3.8 Return Statement

It is possible to early return from subroutines and stop the execution of events with the `return` statement.

Example:

```
var v = 0

sub toto
if v == 0 then
    return
end
v = 1

onevent test
callsub toto
return
v = 2
```

3.8.1 Initialization

Statements placed between the variable declarations and the subroutines and event handlers are run when the program is initialized:

```
var state

state = 0
call leds.bottom.left(0,0,32)
call leds.bottom.right(0,32,0)
call leds.top(32,0,0)
```

While the initialization of `state` could have been done in its declaration, the initialization of the leds must be done by statements. When programming a robot, you will usually want to define some event that will re-initialize the state of the robot. This is possible by writing the statements within a subroutine and calling it from the event handler. It is also possible to call the subroutine as part of the program initialization even though it has not yet been declared:

```
var state

callsub init  # Initialize the program

# Subroutine for initialization
sub init
    state = 0
    call leds.bottom.left(0,0,32)
    call leds.bottom.right(0,32,0)
    call leds.top(32,0,0)

# Re-initialize when center button is touched
onevent button.center
    callsub init
```

3.9 Sending external events

The programme can send external events by using the `emit` keyword, followed by the name of the event and the name of the variable to send, if any. If a variable is provided, the size of the event must match the size of the argument to be emitted. Instead of a variable, array constructors and mathematical expressions can also be used in more complex situations. Events allow the programme to trigger the execution of code on another node or to communicate with an external programme.

```
onevent ir_sensors
emit sensors_values proximity_sensors_values
```

3.10 Native functions

We designed the Aseba language to be simple in order to allow a quick understanding even by novice developers and to implement the virtual machine efficiently on a micro-controller. To perform complex or resource-intensive processing, we provide native functions that are implemented in native code for efficiency. For instance, a native function is the natural way to implement a scalar product.

Native functions are safe, as they specify and check the size of their arguments, which can be constants, variables, array accesses, array constructors and expressions. In the case of an array, you can access the whole array, a single

element, or a sub-range of the array. Native functions take their arguments by reference and can modify their contents but do not return any value. You can use native functions through the `call` keyword.

Example:

```
var a[3] = 1, 2, 3
var b[3] = 2, 3, 4
var c[5] = 5, 10, 15
var d
call math.dot(d, a, b, 3)
call math.dot(d, a, c[0:2], 3)
call math.dot(a[0], c[0:2], 3)
```

3.11 What to read next?

You might be interested to read:

- *Description of the native functions standard library*.

CHAPTER 4

Native functions standard library

4.1 Math

4.1.1 `math.copy(A,B)`

Copy the array B in the array A, element by element: $A_i = B_i$.

4.1.2 `math.fill(A,c)`

Fill each element of the array A with the constant c: $A_i = c$.

4.1.3 `math.addscalar(A, B, c)`

Compute $A_i = B_i + c$ where c is a scalar.

4.1.4 `math.add(A, B, C)`

Compute $A_i = B_i + C_i$ where A, B and C are three arrays of the same size.

4.1.5 `math.sub(A, B, C)`

Compute $A_i = B_i - C_i$ where A, B and C are three arrays of the same size.

4.1.6 `math.mul(A, B, C)`

Compute $A_i = B_i \cdot C_i$ where A, B and C are three arrays of the same size.

Note that this is not a dot product.

4.1.7 math.div(A, B, C)

Compute $A_i = B_i/C_i$ where A, B and C are three arrays of the same size.

An exception will be triggered if a division by zero occurs.

4.1.8 math.min(A, B, C)

Write the minimum of each element of B and C in A where A, B and C are three arrays of the same size: $A_i = \min(B_i, C_i)$.

4.1.9 math.max(A, B, C)

Write the maximum of each element of B and C in A where A, B and C are three arrays of the same size: $A_i = \max(B_i, C_i)$.

4.1.10 math.clamp(A,B,C,D)

Clamp each element B_i and store it in A_i so that $C_i < A_i < D_i$.

4.1.11 math.dot(r, A, B, n)

Compute the dot product between two arrays of the same size A and B: $r = \frac{\sum_i (A_i \cdot B_i)}{2^n}$

4.1.12 math.stat(V, min, max, mean)

Compute the minimum, the maximum and the mean values of array V.

4.1.13 math.argbounds(A, argmin, argmax)

Get the indices *argmin* and *argmax* corresponding to the minimum respectively maximum values of array A.

4.1.14 math.sort(A)

Sort the array A in place.

4.1.15 math.muldiv(A, B, C, D)

Compute multiplication-division using internal 32-bit precision: $A_i = \frac{B_i \cdot C_i}{D_i}$.

An exception will be triggered if a division by zero occurs.

4.1.16 `math.atan2(A, Y, X)`[1]

Compute $A_i = \arctan\left(\frac{Y_i}{X_i}\right)$ using the signs of Y_i and X_i to determine the quadrant of the output, where A, Y and X are three arrays of the same size.

Note that $X_i = 0$ and $Y_i = 0$ will produce $A_i = 0$.

4.1.17 `math.sin(A, B)`[1]

Compute $A_i = \sin(B_i)$ where A and B are two arrays of the same size.

4.1.18 `math.cos(A, B)`[1]

Compute $A_i = \cos(B_i)$ where A and B are two arrays of the same size.

4.1.19 `math.rot2(A, B, angle)`[1]

Rotate the array B by *angle*, write the result to A.

Note that A and B must both be arrays of size 2.

4.1.20 `math.sqrt(A, B)`

Compute $A_i = \sqrt{B_i}$ where A and B are two arrays of the same size.

4.1.21 `math.rand(v)`

Return a random value v in the range $-32768 : 32767$.

Since a scalar is considered to be an array of size one, you can use these functions on scalars:

```
var theta = 16384
var cos_theta
call math.cos(cos_theta, theta)
```

4.2 Double-ended queues

The *Deque* native library provides functions for double-ended queue operations in an object-oriented style on specially-formatted arrays. The array for a *deque* object must be of size 2 + m $\cdot k$ where k is the size of the tuples[2] in the deque, and m is the maximum number of tuples to be stored.

An *index* i into a deque is between two elements: the integer i counts the number of elements to the left of the index.

[1] The trigonometric functions map the angles $[-pi, pi[$ radians to $-32768, 32767$. The resultant sin and cos values are similarly mapped, namely $[-1.0, 1.0[$ to $-32768, 32767$.

[2] A *tuple* is simply a small array of values that are inserted in the *deque* together

4.2.1 deque.size(Q,n)

Set n to the number of elements in deque Q. If n=0 then Q is empty. Note that n must be divided by the tuple size to obtain the number of tuples in Q.

4.2.2 deque.push_front(Q,A)

Insert tuple A before the first tuple of deque Q.

4.2.3 deque.push_back(Q,A)

Insert tuple A after the last tuple in deque Q.

4.2.4 deque.pop_front(Q,A)

Remove the first $length(\mathrm{A})$ elements of deque Q and place them in tuple A. <<<<<<< HEAD

4.2.5 deque.pop_back(Q,A)

Remove the last $length(\mathrm{A})$ elements of deque Q and place them in tuple A.

4.2.6 deque.pop_back(Q,A)

Remove the last $length(\mathrm{A})$ elements of deque Q and place them in tuple A.

>>>>>>> nsis `deque.get(Q,A,i)` ^^^^^^^^^^^^^^^^^^^^^^^^ Copy into tuple A , $length(\mathrm{A})$ elements from deque Q starting from index i.

4.2.7 deque.set(Q,A,i)

Copy into deque Q starting at index i, $length(\mathrm{A})$ elements from tuple A.

4.2.8 deque.insert(Q,A,i)

Shift right the suffix of deque Q starting at index i by $length(\mathrm{A})$ elements, then copy tuple A into the deque Q at that index.

4.2.9 deque.erase(Q,i,k)

Erase k elements from deque Q at index i by shifting the suffix starting from i+k left. Length k should be the tuple size or a multiple.

4.2.10 Example

Here is a simple motion queue, that accepts *operations* defined by a time and motor speeds, and executes them first-in, first-out.

```
var operation[3] # Tuple of size 3
var Queue[2 + (3*40)] # Store up to 40 operation tuples
var n

sub motion_add
    call deque.push_back(Queue, event.args[0:2])

onevent timer0
    call deque.size(Queue, n)
    if n > 0 then
        call deque.pop_front(Queue, operation)
        timer.period[0] = operation[0]
        motor.left.target = operation[1]
        motor.right.target = operation[2]
    end
```

CHAPTER 5

Thymio Programming Interface

This page describes the programming capabilities of Thymio. It lists the different variables and functions and indicates to which elements of the robot they refer.

5.1 Buttons

Thymio holds 5 capacitive buttons corresponding to the arrows and to the central button. Five variables hold the state of these buttons (1 = pressed, 0 = released):

- `button.backward` : backward arrow
- `button.left` : left arrow
- `button.center` : central button
- `button.forward` : forward arrow
- `button.right` : right arrow

Thymio updates this array at a frequency of 20 Hz and generates the `buttons` event after every update. Moreover, for each of these buttons, when it is pressed or released, a corresponding event with the same name is fired.

5.2 Distance sensors

5.2.1 Horizontal

Thymio has 7 distance sensors around its periphery. An array of 7 variables, `prox.horizontal`, holds the values of these sensors:

- `prox.horizontal[0]` : front left
- `prox.horizontal[1]` : front middle-left
- `prox.horizontal[2]` : front middle

- `prox.horizontal[3]` : front middle-right
- `prox.horizontal[4]` : front right
- `prox.horizontal[5]` : back left
- `prox.horizontal[6]` : back right

The values in this array vary from 0 (the robot does not see anything) to several thousand (the robot is very close to an obstacle). Thymio updates this array at a frequency of 10 Hz, and generates the `prox` event after every update.

5.2.2 Ground

Thymio holds 2 ground distance sensors. These sensors are located at the front of the robot. As black grounds appear like no ground at all (black absorbs the infrared light), these sensors can be used to follow a line on the ground. Three arrays hold the values of these sensors:

- `prox.ground.ambiant` : ambient light intensity at the ground, varies between 0 (no light) and 1023 (maximum light)
- `prox.ground.reflected` : amount of light received when the sensor emits infrared, varies between 0 (no reflected light) and 1023 (maximum reflected light)
- `prox.ground.delta` : difference between reflected light and ambient light, linked to the distance and to the ground colour.

For each array, the index 0 corresponds to the left sensor and the index 1 to the right sensor. As with the distance sensors, Thymio updates this array at a frequency of 10 Hz and generates the (same) `prox` event after every update.

5.3 Local communication

Thymio can use its horizontal infrared distance sensors to communicate a value to peer robots within a range of about 15 cm. This value is sent at 10 Hz while processing the distance sensors. Thymio sends an 11-bit value (but future firmware could use one of the bits for internal use, thus it is better to stay within 10 bits). To use the communication, call the `prox.comm.enable(state)` function, with 1 in `state` to enable communication or 0 to turn it off. If the communication is enabled, the value in the `prox.comm.tx` variable is transmitted every 100 ms. When Thymio receives a value, the event `prox.comm` is fired and the value is in the `prox.comm.rx` variable.

5.4 Accelerometer

Thymio contains a 3-axes accelerometer. An array of 3 variables, `acc`, holds the values of the acceleration along these 3 axes:

- `acc[0]` : x-axis (from right to left, positive towards left)
- `acc[1]` : y-axis (from front to back, positive towards the rear)
- `acc[2]` : z-axis (from top to bottom, positive towards ground)

The values in this array vary from -32 to 32, with 1 g (the acceleration of the earth's gravity) corresponding to the value 23. Thymio updates this array at a frequency of 16 Hz, and generates the `acc` event after every update. Moreover, when a shock is detected, a `tap` event is emitted.

5.5 Temperature sensor

The `temperature` variable holds the current temperature in tenths of a degree Celsius. Thymio updates this value at 1 Hz and generates the `temperature` event after every update.

5.6 Timers

Thymio provides two user-defined timers. An array of 2 values, `timer.period`, allows to specify the period of the timers in ms:

- `timer.period[0]` : period of timer 0 in milliseconds
- `timer.period[1]` : period of timer 1 in milliseconds

The timer starts the countdown when it is initialized.

When the period expires, the timer generates a `timer0` respectively `timer1` event. These events are managed in the same way as all the others and cannot interrupt an already executing event handler.

If you restart a program with a timer, the timer could still be counting down and an event can occur before you expect it. This is not usually a problem if you use a timer that expires repeatedly at short intervals, because you can set a state variable to ignore timer events until you are ready. It is recommended that you *not* use the timer for a *single (long) interval* because the results can be inconsistent.

5.7 LEDs

Thymio holds many LEDs spread around its body. Most of them are associated with sensors and can highlight their activations: by default, the intensity of the LED is linked to the sensor value. However, once LEDs are used in the code, the programmer takes over control and they no longer reflect the sensor values.

Native functions allow the various LEDs to be controlled. For all LEDs, their intensity values range from 0 (off) to 32 (fully lit).

5.7.1 The LED circle on top of the robot

8 yellow LEDs make up a circle on top of the robot, around the buttons.

Default activation: reflects the values of the accelerometer. All LEDs are off when the robot is horizontal. When the robot tilts, a single LED shows the lowest point, with an intensity proportional to the tilt angle.

- `leds.circle(led 0, led 1, led 2, led 3, led 4, led 5, led 6, led 7)` where `led 0` sets the intensity of the LED at the front of the robot, the others are numbered clockwise.

5.7.2 The RGB LEDs

There are two RGB LEDs on the top of robot, driven together. These are the LEDs that show the behaviour of the robot. There are two other RGB LEDs on the bottom of the robot, which can be driven separately.

Default activation: off when in Aseba mode.

- `leds.top(red, green, blue)` sets the intensities of the top LEDs.
- `leds.bottom.left(red, green, blue)` sets the intensities of the bottom-left LED.

- `leds.bottom.right(red, green, blue)` sets the intensities of the bottom-right LED.

5.7.3 The LEDs of proximity sensors

Every proximity sensor has a companion red LED on its side (the front sensor has two LEDs, one on each side).

Default activation: on when an object is close to the associated sensor, with an intensity inversely proportional to the distance.

- `leds.prox.h(led 1, led 2, led 3, led 4, led 5, led 6, led 7, led 8)` sets the LEDs of the front and back horizontal sensors. `led 1` to `led 6` correspond to the front LEDs, from left to right, while `led 7` and `led 8` correspond to the left and right back LEDs.
- `leds.prox.v(led 1, led 2)` sets the LEDs associated with the bottom sensors, left and right.

5.7.4 The Button LEDs

Four red LEDs are placed between the buttons.

Default activation: For each arrow button, one LED lights up when it is pressed. When the centre button is pressed, all four LEDs light up.

- `leds.buttons(led 1, led 2, led 3, led 4)` control these LEDs, with `led 1` corresponding to the front LED, then clockwise numbering.

5.7.5 The LED of the RC receiver

This red LED is located close to the remote-control (infrared) receiver.

Default activation: blinks when the robot receives an RC5 code.

- `leds.rc(led)` controls this LED.

5.7.6 The LEDs of the temperature sensor

These two LEDs (one red and one blue) are located close to the temperature sensor.

Default activation: red if the temperature is over 28°C, red and blue between 28° and 15°, blue if the temperature is below 15°.

- `leds.temperature(red, blue)` controls this LED.

5.7.7 The microphone LED

This blue LED is located close to the microphone.

Default activation: off.

- `leds.sound(led)` controls this LED.

There are also other LEDs that the user cannot control:

- 3 green LEDs on the top of the robot show the battery voltage
- a blue and a red LED on the back of the robot show the charge status
- a red LED on the back of the robot shows the SD-card status

5.8 Motors

You can change the wheel speeds by writing in these variables:

- `motor.left.target`: requested speed for left wheel
- `motor.right.target`: requested speed for right wheel

You can read the real wheel speeds from these variables:

- `motor.left.speed` : real speed of left wheel
- `motor.right.speed` : real speed of right wheel

The values range from -500 to 500. A value of 500 approximately corresponds to a linear speed of 20 cm/s. You can read the value of the motor commands from the variables `motor.left.pwm` and `motor.right.pwm`.

5.9 Sound

5.9.1 Sound-intensity detection

The Thymio can detect when the ambient sound is above a given intensity and emit an event.
The variable `mic.intensity` shows the current microphone intensity (in the range 0 to 255), while variable `mic.threshold` contains the limit intensity for the event. If `mic.intensity` is above `mic.threshold`, then the event `mic` is generated.

5.9.2 Playing and recording sounds

You can play synthetic or system sounds. Moreover, if you have installed a micro-SD card formatted as FAT, you can record and play your own sounds. The files are stored in the micro-SD card, in wave format, 8-bit unsigned, 8 kHz. When Thymio finishes playing a sound requested through Aseba, it fires the event `sound.finished`. It does not fire an event if playing is interrupted or if a new sound is played.

5.9.3 Synthetic sound

The native function `sound.freq` plays a frequency, specified in Hz, for a certain duration, specified in 1/60 s. Specifying a 0 duration plays the sound continuously and specifying a -1 duration stops the sound.

5.9.4 Changing the primary wave

Synthetic sound generation works by re-sampling a primary wave. By default, it is a triangular wave, but you can define your own wave using the `sound.wave` native function. This function takes as input an array of 142 samples, with values from -128 to 127. This buffer should represent one wave of the tonic frequency specified in `sound.freq`. As Thymio plays sounds at 7812.5 Hz, this array is played completely at the frequency of 7812.5/142 = ~55 Hz. Playing a sound of a higher frequency skips samples in the array.

5.9.5 Recording

You can record sounds using the `sound.record` native function. This function takes as parameter a record number from 0 to 32767. Files are stored on the micro-SD card under the name `Rx.wav` where `x` is the parameter passed to the `sound.record` function. To stop a recording, call the `sound.record` function with the value of -1.

5.9.6 Replaying

You can replay a recorded sound using the `sound.replay` native function. This function takes as parameter a record number from 0 to 32767 and will replay file `Rx.wav` from the SD card where `x` is the parameter passed to the `sound.replay` function. To stop a replay, call the `sound.replay` function with the value of -1.

5.9.7 Duration (from firmware version 11)

You can retrieve the duration of a recorded sound using the `sound.duration(x,duration)` native function. Its first parameter, `x`, is a number from 0 to 32767 which is the index of file `Rx.wav` from the SD card. The result in 1/10 of seconds is put in the variable `duration` as second parameter.

5.9.8 Creating sound on your computer

You can create sounds for Thymio using your computer. An efficient way to do so is to use the Audacity software, version 1.3, which exists for various operating systems. Here are the steps to create a sound compatible with the Thymio:

- Once Audacity has started, change the *project rate* from 44100 Hz (default) to 8000 Hz. This setting is located at the bottom-left of Audacity's window.
- Record your sound with the red record key in the top-left part of the window. You should see the cursor advancing and the wave changing. Stop with the stop button.
- Your sound should be in mono (Tracks->Stereo to Mono)
- Go to the *File* menu under *Export...*
- Give a file name, for instance `P0.wav` for the first file to play using the `sound.play` native function.
- Choose *other uncompressed files* as format *format*.
- Under *options*, choose a *WAV (Microsoft)* header and as *Encoding*, choose *Unsigned 8 bit PCM*.
- Make sure that no metadata values ares set.
- Save or copy the file to the micro-SD card.

Here's an instructional video on how to do the above.

5.9.9 Play

You can play a user-defined sound using the `sound.play` native function, which takes a record number from 0 to 32767 as parameter. The file must be available on the micro-SD card under the name `Px.wav` where `x` is the parameter passed to the `sound.play` function. To stop playing a sound, call the `sound.play` function with the value -1.

5.9.10 System sound

You can play a system sound using the `sound.system` native function, which takes a record number from 0 to 32767 as parameter. Some sounds are available in the firmware (see below), but you can overwrite these sounds and add new ones using the SD-card. In this case, the file must be named `Sx.wav` where `x` is the parameter passed to the `sound.system` function. To stop playing a sound, call the `sound.system` function with the value -1.

5.9.11 System sound library

The following sounds are available:

parameter	description
-1	stop playing sound
0	startup sound
1	shutdown sound (this sound is not reconfigurable)
2	arrow button sound
3	central button sound
4	free-fall (scary) sound
5	collision sound
6	target ok for friendly behaviour
7	target detect for friendly behaviour

5.10 Remote control

Thymio contains a receiver for infrared remote controls compatible with the RC5 protocol. When Thymio receives an RC5 code, it generates the `rc5` event. In this case, the variables `rc5.address` and `rc5.command` are updated.

5.11 Read and write data from the SD card

If an SD card is present, the variable `sd.present` is set to 1 (otherwise 0), and Thymio can read and write data to files. Only a single file can be open at any given time. The unit of reading/writing is a signed 16-bit binary value. The functions provided are:

- `sd.open(x,status)`: opens the file `Ux.DAT`. The value `x` should be a number between [0:32767], using -1 closes the currently open file. A value of 0 is written in the `status` variable if the operation was successful, -1 if the operation has failed.
- `sd.write(data,written)`: attempts to write the complete `data` array in the currently opened file. The number of values written is returned in the `written` parameter. It should be equal to the size of `data`, except if the card was full, or if the file was larger than 4 Gb, or no file was open.
- `sd.read(data,read)`: reads and fills the `data` array from the currently opened file. The number of values read is returned in the `read` parameter. It should be equal to the size of `data`, except when the end of the file is encountered or no file was open.
- `sd.seek(position,status)`: moves the current read and write pointers in the currently opened file. The cursor is moved to the absolute `position` in the opened file. The valid range is [0:65535]. It is currently not possible to seek to a position after 65535. A value of 0 is written in the `status` variable if the operation was successful, -1 if the operation has failed.

The format consist of a simple concatenation of the signed 16-bit binary values.

Note: do not remove the SD card while the robot is turned on. Always power-off the robot before removing the SD card.

5.12 Loading a program from the SD card

Thymio can load a program from the SD card. When it boots, Thymio loads the file `vmcode.abo` from the SD card if present.

To obtain the `vmcode.abo` file from your .aesl file, open Aseba Studio and open your program (let's call it `myprogram.aesl`). Then click on (**1**) "Tool", then (**2**) "Save binary code...", then (**3**) "...of thymio". You will see a dialog box opening (**4**). Choose a place where to save your file and that's it, you saved `myprogram.aesl` with the .abo format. Don't forget to call it `vmcode.abo` if you want your Thymio to read it when it starts.

5.13 Table of local events

event	description	frequency (Hz)	result
`button.backward`	back arrow was pressed or released	upon action	`button.backward`
`button.left`	left arrow was pressed or released	upon action	`button.left`
`button.center`	central button was pressed or released	upon action	`button.center`
`button.forward`	front arrow was pressed or released	upon action	`button.forward`
`button.right`	right arrow was pressed or released	upon action	`button.right`
`buttons`	button values have been probed	50	`buttons.backward`, `buttons.left`, `buttons.center`, `buttons.forward`, `buttons.right`
`prox`	proximity sensors were read	10	`prox.horizontal[0-7]`, `prox.ground.ambiant[0-1]`, `prox.ground.reflected[0-1]` and `prox.ground.delta[0-1]`
`prox.comm`	value received from IR sensors	upon value reception	`prox.comm.rx`
`tap`	a shock was detected	upon shock	`acc[0-2]`
`acc`	the accelerometer was read	16	`acc[0-2]`
`mic`	ambient sound intensity was above threshold	when condition is true	`mic.intensity`
`sound.finished`	a sound started by aseba has finished playing by itself	when sound finishes	
`temperature`	temperature was read	1	`temperature`
`rc5`	the infrared remote-control receiver got a signal	upon signal reception	`rc5.address` and `rc5.command`
`motor`	PID is executed	100	`motor.left/right.speed`, `motor.left/right.pwm`
`timer0`	when timer 0 period expires	user-defined	
`timer1`	when timer 1 period expires	user-defined	

CHAPTER 6

Simulated Thymio API

This page describe the APIs, methods and VPL blocks available when programming a simulated Thymio. Because the simulated Thymio is in a limited, virtual environment, not all features are implemented. However, in order to be compatible with the real Thymio, all variables, functions and events are available.

If you use them, you will not get a compilation error, however some events might never happen, some functions might do nothing and some variables might never change.

6.1 Aseba

6.1.1 Variables

Variable	Status
`button.backward`, `button.left`, `button.center`, `button.forward`, `button.right`	simulated, 0 by default, 1 when the respective buttons are clicked on with the mouse
`prox.horizontal[0-6]`	simulated, return values depending on what is around the robot in the simulator with the addition of noise
`prox.ground.ambiant[0-1]`	**always 0**
`prox.ground.reflected[0-1]`	Simulated, return values depending on the colour of the ground with the addition of noise. The simulated environment is 2-dimensional, there can be no holes under Thymio and Thymio cannot be tilted or laid on one side)
`prox.ground.delta[0-1]`	simulated, returns the same as `prox.ground.reflected[0-1]`
`prox.comm.rx`, `prox.comm.tx`	**always 0**
`acc[0-2]`	**always 0**
`temperature`	**always 220**
`motor.left.target`, `motor.right.target`	default 0, can be defined by the user to set the motors' speeds. The simulated Thymio can move on a plane.
`motor.left.speed`, `motor.right.speed`	returns `motor.left.target`, `motor.right.target`
`motor.left.pwm`, `motor.right.pwm`	**always 0**
`rc5.address`	**always 0**
`rc5.command`	**always 0**
`mic.intensity`	**always 0**
`mic.threshold`	**can be defined by the user but will have no effect**
`timer.period[0]`, `timer.period[1]`	can be defined by the user to setup the `timer0` and `timer1` events

6.1.2 Functions

Function	Status
`prox.comm.enable(state)`	**does nothing**

`leds.circle(l0, l1, l2, l3, l4, l5, l6, l7)`
`leds.top(r, g, b)`
`leds.bottom.left(r, g, b)`
`leds.bottom.right(r, g, b)`
`leds.buttons(l0, l1, l2, l3)`
`leds.prox.h(l0, l1, l2, l3, l4, l5, l6, l7)`
	These LEDs are displayed as textures on the simulated robot

`leds.rc(l)`
`leds.temperature(r, b)`
`leds.sound(l)`
`leds.prox.v(l0, l1)`
	do nothing

`sound.freq`
`sound.wave`
`sound.record`
`sound.replay`
`sound.play`
	do nothing

`sd.open(x,status)`
`sd.write(data,written)`
`sd.read(data,read)`
`sd.seek(position,status)`
	Simulated

6.1.3 Events

Event	Status
`button.backward`, `button.left`, `button.forward`, `button.right`, `button.center`	triggered when the user clicks with the mouse on one of the simulated Thymio's buttons
`buttons`	regular, 50 Hz
`prox`	regular, 10 Hz
`prox.comm`	**never triggered**
`tap`	triggered upon shock in the simulator or when the user clicks with the mouse on the simulated Thymio (but not on the buttons)
`acc`	regular, 16 Hz
`mic`	**never triggered**
`sound.finished`	**never triggered**
`temperature`	regular, 1Hz
`rc5`	**never triggered**
`motor`	regular, 100 Hz
`timer0`	triggered when timer 0 period expires, user-defined
`timer1`	triggered when timer 1 period expires, user-defined

6.2 VPL Blocks

VPL Block	Status
	implemented, triggered when the user clicks with the mouse on one of the simulated Thymio's buttons
	implemented, detects objects in the simulator
	implemented, detects ground colour in the simulator
	implemented, triggered upon shock in the simulator or when the user clicks with the mouse on the simulated Thymio (but not on the buttons)
	not implemented, never triggered
	implemented, the simulated Thymio can move on a plane
	implemented, displayed as textures on the simulated robot
	implemented, displayed as textures on the simulated robot
	not implemented, does nothing
	implemented, triggered when timer expires
	implemented
	implemented
	implemented, detects objects in the simulator
	implemented, detects ground colour in the simulator
	Thymio cannot be tilted in the simulator
	Thymio cannot be tilted in the simulator

CHAPTER 7

Building Aseba

7.1 Requierements & Dependencies

Aseba requieres a C++14 compatible compiler. We recommand `GCC 6`, `Clang 5` or `MSVC 19` (Visual Studio 17).

Aseba depends on Qt5.9 or greater. You will also need `cmake` 3.1 or greater, we recommend you use the latest version available.

7.2 Getting the source code

The source code of Aseba is hosted on github. To fetch the source, you must first install Git .

Then, to clone aseba, execute:

```
git clone --recursive https://github.com/mobsya/aseba.git
cd aseba
```

All the commands given in the rest of this document assume the current path is the root folder of the cloned repository.

7.3 Getting Started on Windows with MSVC

Aseba Builds on Windows Vista or greater.

- Install `Bonjour`. You will find the installer in `third_party/bonjour/bonjoursdksetup.exe`
- Install Qt from Qt's website https://download.qt.io/official_releases/qt/. Once installed, th `bin` repertory of Qt must be in your path.
- Install Visual Studio
- Install `CMake`

In a developer prompt run:

```
mkdir build;
cd build
cmake -DBUILD\_SHARED\_LIBS=OFF -DCMAKE\_BUILD\_TYPE=Release -DCMAKE\_PREFIX\_PATH=";
↪" ..
nmake
```

7.3.1 Additional Dependencies

Aseba can be improved with several other dependencies * SDL (Joystick Support) * QWT (Charts) * libxml2

7.4 Getting Started on Windows with MingW

An easy way to compile Aseba on Windows is to use msys2.

7.4.1 Preliminary: download, install and update msys2

Download and install msys2 from www.msys2.org. If you have a 64-bit version of Windows, take the `x86_64` installer, otherwise the `i686` one.

Once msys2 is installed, start the shell by running the `MSYS2 MSYS` application. In the shell, update msys2 by typing:

```
pacman -Syu
```

Then restart the shell and update the packages by typing:

```
pacman -Su
```

7.4.2 Install the dependencies

In the msys2 shell, install the dependencies by typing:

```
pacman -S mingw-w64-i686-{toolchain,cmake,qt5,qwt-qt5,libxml2,SDL2} git make
```

If you want to build Windows 64 binaries, replace `i686` by `x86_64`

7.4.3 Building Aseba

In a msys2 prompt, run

```
mkdir build && cd build
cmake -DBUILD_SHARED_LIBS=OFF -DCMAKE_BUILD_TYPE=Release -DCMAKE_PREFIX_PATH="<path␣
↪of qt>;<path of bonjour>" ..
make
```

7.5 Getting Started on OSX

You will need OSX 10.11 or greater

- Install Homebrew.
- In the cloned repository run

```
brew update brew tap homebrew/bundle brew bundle
```

Then you can create a build directory and build Aseba

```
mkdir build && cd build
cmake -DCMAKE_BUILD_TYPE=Release -DBUILD_SHARED_LIBS=OFF ..
make
```

7.6 Getting Started on Linux

7.6.1 Dependencies On Ubuntu & Debian

```
sudo apt-get install qttools5-dev-tools \
                     qtbase5-dev \
                     qttools5-dev \
                     libqt5help5 \
                     qt5-qmake \
                     libqt5opengl5-dev \
                     libqt5svg5-dev \
                     libqt5x11extras5-dev \
                     libqwt-qt5-dev \
                     libudev-dev \
                     libxml2-dev \
                     libsdl2-dev \
                     libavahi-compat-libdnssd-dev \
                     cmake \
                     g++ \
                     git \
                     make \
```

7.6.2 Building Aseba

```
mkdir build && cd build
cmake -DCMAKE_BUILD_TYPE=Release -DBUILD_SHARED_LIBS=OFF ..
make
```

7.6.3 A note about permissions

If you will be connecting to your robot through a serial port, you might need to add yourself to the group that has permission for that port. In many distributions, this is the "dialout" group and you can add yourself to that group and use the associated permissions by running the following commands:

```
sudo usermod -a -G dialout $USER
newgrp dialout
```

7.7 Getting Started on Android

VPL 2 can be build for Android. Other tools such as studio, playground and the old VPL are not compatible with android.

To build the android version you will need:

- The android tools for your system
- The android NDK - tested with version 10 - currently not compatible wuth newer NDK
- Qt 5.10 for android - which you can install through the Qt installer
- CMake 3.7 or greater

7.7.1 Building VPL 2

First, you need to prepare some environment variables

```
export ANDROID_SDK=<path_of_the_android_sdk>
export ANDROID_NDK=<path_of_the_android_ndk>
export CMAKE_PREFIX_PATH="${CMAKE_PREFIX_PATH}:$HOME/<path_of_qt5_for_android>"
```

Then you can build vpl2 with cmake. An APK will be generated in `build/bin`

```
mkdir build && cd build
cmake -DCMAKE_BUILD_TYPE=Release -DANDROID_NATIVE_API_LEVEL=14 -DANDROID_STL=gnustl_
↪shared -DCMAKE_TOOLCHAIN_FILE=`pwd`/../android/qt-android-cmake/toolchain/android.
↪toolchain.cmake
make
```

7.8 Advanced Setup

7.8.1 Running tests

Once the build is complete, you can run `ctest` in the build directory to run the tests.

7.8.2 Ninja

The compilation of Aseba can be significantly speed up using `ninja` insteadf of make. Refer to the documentation of `cmake` and `ninja`.

CHAPTER 8

C++ Reference

Generated from doxygen

struct *Aseba***AbstractTreeNode** : **public** *Aseba*::*Node*

#include <tree.h> Virtual class for abstraction nodes abort() if you try to *typeCheck()*, *optimize()* or *emit()* to force correct expansion into regular Aseba nodes.

Subclassed by *Aseba::ArithmeticAssignmentNode*, *Aseba::MemoryVectorNode*, *Aseba::TupleVectorNode*, *Aseba::UnaryArithmeticAssignmentNode*

Public Functions

Aseba::*AbstractTreeNode***AbstractTreeNode**(**const** *SourcePos* &*sourcePos*)

Constructor.

Node **Aseba*::*AbstractTreeNode***expandAbstractNodes**(std::wostream **dump*) = 0

Second pass to expand "abstract" nodes into more concrete ones.

Generically traverse the tree, expand the children, and perform garbagge collection.

ReturnType *Aseba*::*AbstractTreeNode***typeCheck**(*Compiler* **compiler*)

Typecheck this node, throw an exception if there is any type violation.

Node **Aseba*::*AbstractTreeNode***optimize**(std::wostream **dump*)

Optimize this node, return the optimized node.

unsigned *Aseba*::*AbstractTreeNode***getStackDepth**() **const**

Return the stack depth requirement for this node and its children.

void *Aseba*::*AbstractTreeNode***emit**(*PreLinkBytecode* &*bytecodes*) **const**

Generate bytecode.

struct *Aseba***ArithmeticAssignmentNode** : **public** *Aseba*::*AbstractTreeNode*

#include <tree.h> *Node* for operations like "vector (op)= something" Will expand to "vector = vector (op) something" children[0] is a *MemoryVectorNode* children[1] is whatever *Node* for the right operand.

Public Functions

Aseba::*ArithmeticAssignmentNode***ArithmeticAssignmentNode**(**const** *SourcePos* &*sourcePos*, AsebaBinaryOperator *op*, *Node* *left, *Node* *right)
Constructor.

ArithmeticAssignmentNode **Aseba*::*ArithmeticAssignmentNode***shallowCopy**() **const**
Return a shallow copy of the object (children point to the same objects)

void *Aseba*::*ArithmeticAssignmentNode***checkVectorSize**() **const**
Check the consistency in vectors' size.

Node **Aseba*::*ArithmeticAssignmentNode***expandAbstractNodes**(std::wostream *dump)
Expand "left (op)= right" to "left = left (op) right".

Node **Aseba*::*ArithmeticAssignmentNode***expandVectorialNodes**(std::wostream *dump, *Compiler* *compiler = nullptr, unsigned int *index* = 0)
Third pass to expand vectorial operations into mutliple scalar ones.

Generic implementation for non-vectorial nodes.

std::wstring *Aseba*::*ArithmeticAssignmentNode***toWString**() **const**
Return a string representation of this node.

std::wstring *Aseba*::*ArithmeticAssignmentNode***toNodeName**() **const**
Return a string representation of the name of this node.

Public Members

AsebaBinaryOperator *Aseba*::*ArithmeticAssignmentNode***op**
operator

Public Static Functions

ArithmeticAssignmentNode **Aseba*::*ArithmeticAssignmentNode***fromArithmeticAssignmentToken**(**const** *SourcePos* &*sourcePos*, *Compiler*::*Token*::*Type* *token*, *Node* *left, *Node* *right)

Protected Static Functions

AsebaBinaryOperator *Aseba*::*ArithmeticAssignmentNode*`getBinaryOperator`(*Compiler*::*Token*::*Type token*)

Protected Static Attributes

ASEBA_OP_ADD, ASEBA_OP_SUB, ASEBA_OP_MULT, ASEBA_OP_DIV, ASEBA_OP_MOD, ASEBA_OP_SHIFT_LEFT, ASEBA_OP_SHIFT_RIGHT, ASEBA_OP_BIT_OR, ASEBA_OP_BIT_XOR, ASEBA_OP_BIT_AND }]

`struct` *Aseba*`ArrayReadNode` : `public` *Aseba*::*Node*

#include <*tree.h*> *Node* for reading from an array.

children[0] is the index in the array

Public Functions

Aseba::*ArrayReadNode*`ArrayReadNode`(`const` *SourcePos* &*sourcePos*, unsigned *arrayAddr*, unsigned *arraySize*, std::wstring *arrayName*)

Constructor.

ArrayReadNode **Aseba*::*ArrayReadNode*`shallowCopy`() `const`

Return a shallow copy of the object (children point to the same objects)

ReturnType *Aseba*::*ArrayReadNode*`typeCheck`(*Compiler* **compiler*)

Typecheck this node, throw an exception if there is any type violation.

Node **Aseba*::*ArrayReadNode*`optimize`(std::wostream **dump*)

Optimize this node, return the optimized node.

void *Aseba*::*ArrayReadNode*`emit`(*PreLinkBytecode* &*bytecodes*) `const`

Generate bytecode.

std::wstring *Aseba*::*ArrayReadNode*`toWString`() `const`

Return a string representation of this node.

std::wstring *Aseba*::*ArrayReadNode*`toNodeName`() `const`

Return a string representation of the name of this node.

unsigned *Aseba*::*ArrayReadNode*`getVectorAddr`() `const`

return the address of the left-most operand, or E_NOVAL if none

unsigned *Aseba*::*ArrayReadNode*`getVectorSize`() `const`

return the children's size, check for equal size, or E_NOVAL if no child

Public Members

unsigned *Aseba*::*ArrayReadNode*`arrayAddr`

address of the first element of the array

unsigned *Aseba*::*ArrayReadNode*`arraySize`

size of the array, might be used to assert compile-time access checks

std::wstring *Aseba*::*ArrayReadNode*`arrayName`
name of the array (for debug)

`struct` *Aseba*`ArrayWriteNode` : `public` *Aseba*::*Node*
#include <tree.h> *Node* for writing to an array.

Value to store is supposed to be on the stack already children[0] is the index in the array

Public Functions

Aseba::*ArrayWriteNode*`ArrayWriteNode`(`const` *SourcePos* &*sourcePos*, unsigned *arrayAddr*, unsigned *arraySize*, std::wstring *arrayName*)
Constructor.

ArrayWriteNode **Aseba*::*ArrayWriteNode*`shallowCopy`() `const`
Return a shallow copy of the object (children point to the same objects)

ReturnType *Aseba*::*ArrayWriteNode*`typeCheck`(*Compiler* **compiler*)
Typecheck this node, throw an exception if there is any type violation.

Node **Aseba*::*ArrayWriteNode*`optimize`(std::wostream **dump*)
Optimize this node, return the optimized node.

void *Aseba*::*ArrayWriteNode*`emit`(*PreLinkBytecode* &*bytecodes*) `const`
Generate bytecode.

std::wstring *Aseba*::*ArrayWriteNode*`toWString`() `const`
Return a string representation of this node.

std::wstring *Aseba*::*ArrayWriteNode*`toNodeName`() `const`
Return a string representation of the name of this node.

unsigned *Aseba*::*ArrayWriteNode*`getVectorAddr`() `const`
return the address of the left-most operand, or E_NOVAL if none

unsigned *Aseba*::*ArrayWriteNode*`getVectorSize`() `const`
return the children's size, check for equal size, or E_NOVAL if no child

Public Members

unsigned *Aseba*::*ArrayWriteNode*`arrayAddr`
address of the first element of the array

unsigned *Aseba*::*ArrayWriteNode*`arraySize`
size of the array, might be used to assert compile-time access checks

std::wstring *Aseba*::*ArrayWriteNode*`arrayName`
name of the array (for debug)

`struct AsebaLocalEventDescription`
#include <natives.h> Description of a local event.

Public Members

const char *AsebaLocalEventDescription**name**
name of the function

const char *AsebaLocalEventDescription**doc**
documentation of the local event

struct AsebaNativeFunctionArgumentDescription
#include <*natives.h*> Description of an argument of a native function.

Public Members

int16_t AsebaNativeFunctionArgumentDescription**size**
size of the argument in number of values; if negative, template parameter

const char *AsebaNativeFunctionArgumentDescription**name**
name of the argument

struct AsebaNativeFunctionDescription
#include <*natives.h*> Description of a native function.

Public Members

const char *AsebaNativeFunctionDescription**name**
name of the function

const char *AsebaNativeFunctionDescription**doc**
documentation of the function

AsebaNativeFunctionArgumentDescription AsebaNativeFunctionDescription**arguments**[]
arguments, terminated by an element of size 0

struct AsebaVariableDescription
#include <*natives.h*> Description of a variable.

Public Members

uint16_t AsebaVariableDescription**size**
size of the variable

const char *AsebaVariableDescription**name**
name of the variable

struct AsebaVMDescription
#include <*natives.h*> Description of all variable.

Public Members

const char *AsebaVMDescription**name**
name of the microcontroller

AsebaVariableDescription AsebaVMDescription**variables**[]
named variables, terminated by a variable of size 0

struct AsebaVMState

#include <*vm.h*> This structure contains the state of the Aseba VM.

This is the required and the sufficient data for the VM to run. This is not sufficient for the compiler to build bytecode, as there is no description of variable names and sizes nor description of native function. For this, a description corresponding to the actual target must be provided to the compiler. ALL fields of this structure have to be initialized correctly for aseba to work. An initial call to AsebaVMInitStep must be done prior to any call to AsebaVMPeriodicStep or AsebaVMEventStep.

Public Members

uint16_t *AsebaVMState*::**nodeId**

uint16_t *AsebaVMState*::**bytecodeSize**
total amount of bytecode space

uint16_t **AsebaVMState*::**bytecode**
bytecode space of size bytecodeSize

uint16_t *AsebaVMState*::**variablesSize**
total amount of variables space

int16_t **AsebaVMState*::**variables**
variables of size variableCount

uint16_t *AsebaVMState*::**stackSize**
depth of execution stack

int16_t **AsebaVMState*::**stack**
execution stack of size stackSize

uint16_t *AsebaVMState*::**flags**

uint16_t *AsebaVMState*::**pc**

int16_t *AsebaVMState*::**sp**

uint16_t *AsebaVMState*::**breakpoints**[**ASEBA_MAX_BREAKPOINTS**]

uint16_t *AsebaVMState*::**breakpointsCount**

struct *Aseba*::**AssignmentNode** : **public** *Aseba*::*Node*

#include <*tree.h*> *Node* for assignation.

children[0] is store code children[1] expression to store

Public Functions

Aseba::*AssignmentNode*::**AssignmentNode**(**const** *SourcePos* &*sourcePos*)
Constructor.

Aseba::*AssignmentNode*::**AssignmentNode**(**const** *SourcePos* &*sourcePos*, *Node* **left*, *Node* **right*)
Constructor.

AssignmentNode **Aseba*::*AssignmentNode*::**shallowCopy**() **const**
Return a shallow copy of the object (children point to the same objects)

void *Aseba*::*AssignmentNode*::**checkVectorSize**() **const**
Check the consistency in vectors' size.

Node *Aseba*::*AssignmentNode***expandVectorialNodes**(std::wostream *dump, *Compiler* *compiler = nullptr, unsigned int *index* = 0)
Assignment between vectors is expanded into multiple scalar assignments.

Node::*ReturnType* *Aseba*::*AssignmentNode***typeCheck**(*Compiler* *compiler)
Typecheck this node, throw an exception if there is any type violation.

Node *Aseba*::*AssignmentNode***optimize**(std::wostream *dump)
Optimize this node, return the optimized node.

void *Aseba*::*AssignmentNode***emit**(*PreLinkBytecode* &bytecodes) **const**
Generate bytecode.

std::wstring *Aseba*::*AssignmentNode***toWString**() **const**
Return a string representation of this node.

std::wstring *Aseba*::*AssignmentNode***toNodeName**() **const**
Return a string representation of the name of this node.

struct *Aseba***BinaryArithmeticNode** : **public** *Aseba*::*Node*
#include <tree.h> *Node* for binary arithmetic.

children[0] is left expression children[1] is right expression

Public Functions

Aseba::*BinaryArithmeticNode***BinaryArithmeticNode**(**const** *SourcePos* &*sourcePos*)

Aseba::*BinaryArithmeticNode***BinaryArithmeticNode**(**const** *SourcePos* &*sourcePos*, AsebaBinaryOperator *op*, *Node* *left, *Node* *right)
Constructor.

BinaryArithmeticNode *Aseba*::*BinaryArithmeticNode***shallowCopy**() **const**
Return a shallow copy of the object (children point to the same objects)

void *Aseba*::*BinaryArithmeticNode***deMorganNotRemoval**()
Recursively apply de Morgan law as long as nodes are logic operations, and then invert comparisons.

Node::*ReturnType* *Aseba*::*BinaryArithmeticNode***typeCheck**(*Compiler* *compiler)
Typecheck this node, throw an exception if there is any type violation.

Node *Aseba*::*BinaryArithmeticNode***optimize**(std::wostream *dump)
Optimize this node, return the optimized node.

unsigned *Aseba*::*BinaryArithmeticNode***getStackDepth**() **const**
Return the stack depth requirement for this node and its children.

void *Aseba*::*BinaryArithmeticNode***emit**(*PreLinkBytecode* &bytecodes) **const**
Generate bytecode.

std::wstring *Aseba*::*BinaryArithmeticNode***toWString**() **const**
Return a string representation of this node.

std::wstring *Aseba*::*BinaryArithmeticNode***toNodeName**() **const**
Return a string representation of the name of this node.

Public Members

AsebaBinaryOperator *Aseba*::*BinaryArithmeticNode***op**
operator

Public Static Functions

BinaryArithmeticNode **Aseba*::*BinaryArithmeticNode***fromComparison**(**const** *SourcePos* &*sourcePos*, *Compiler*::*Token*::*Type op*, *Node* **left*, *Node* **right*)
Create a binary arithmetic node for comparaison operation op.

BinaryArithmeticNode **Aseba*::*BinaryArithmeticNode***fromShiftExpression**(**const** *SourcePos* &*sourcePos*, *Compiler*::*Token*::*Type op*, *Node* **left*, *Node* **right*)
Create a binary arithmetic node for shift operation op.

BinaryArithmeticNode **Aseba*::*BinaryArithmeticNode***fromAddExpression**(**const** *SourcePos* &*sourcePos*, *Compiler*::*Token*::*Type op*, *Node* **left*, *Node* **right*)
Create a binary arithmetic node for add/sub operation op.

BinaryArithmeticNode **Aseba*::*BinaryArithmeticNode***fromMultExpression**(**const** *SourcePos* &*sourcePos*, *Compiler*::*Token*::*Type op*, *Node* **left*, *Node* **right*)
Create a binary arithmetic node for mult/div/mod operation op.

BinaryArithmeticNode **Aseba*::*BinaryArithmeticNode***fromBinaryExpression**(**const** *SourcePos* &*sourcePos*, *Compiler*::*Token*::*Type op*, *Node* **left*, *Node* **right*)
Create a binary arithmetic node for or/xor/and operation op.

struct *Aseba***BlockNode** : **public** *Aseba*::*Node*
#include <tree.h> *Node* for L"block", i.e. a vector of statements.

Subclassed by *Aseba::ProgramNode*

Public Functions

Aseba::*BlockNode***BlockNode**(**const** *SourcePos* &*sourcePos*)
Constructor.

BlockNode **Aseba*::*BlockNode***shallowCopy**() **const**
Return a shallow copy of the object (children point to the same objects)

Node **Aseba*::*BlockNode***optimize**(std::wostream **dump*)
Optimize this node, return the optimized node.

void *Aseba*::*BlockNode*`emit`(*PreLinkBytecode* &*bytecodes*) `const`
Generate bytecode.

std::wstring *Aseba*::*BlockNode*`toWString`() `const`
Return a string representation of this node.

std::wstring *Aseba*::*BlockNode*`toNodeName`() `const`
Return a string representation of the name of this node.

`struct` *Aseba*`BytecodeElement`
#include <*compiler.h*> A bytecode element.

Public Functions

Aseba::*BytecodeElement*`BytecodeElement`()

Aseba::*BytecodeElement*`BytecodeElement`(unsigned short *bytecode*)

Aseba::*BytecodeElement*`BytecodeElement`(unsigned short *bytecode*, unsigned short *line*)

Aseba::*BytecodeElement*`operator unsigned short`() `const`

unsigned *Aseba*::*BytecodeElement*`getWordSize`() `const`
Return the number of words this element takes in memory.

Public Members

unsigned short *Aseba*::*BytecodeElement*`bytecode` = {0xffff}

unsigned short *Aseba*::*BytecodeElement*`line` = {0}
bytecode itself

line in source code

`struct` *Aseba*`BytecodeVector` : `public` std::deque<*BytecodeElement*>
#include <*compiler.h*> Bytecode array in the form of a dequeue, for construction.

Public Types

`typedef` std::map<unsigned, unsigned> *Aseba*::*BytecodeVector*`EventAddressesToIdsMap`
A map of event addresses to identifiers.

Public Functions

Aseba::*BytecodeVector*`BytecodeVector`()
Constructor.

void *Aseba*::*BytecodeVector*`push_back`(`const` *BytecodeElement* &*be*)

void *Aseba*::*BytecodeVector*`changeStopToRetSub`()
Change “stop” bytecode to “return from subroutine”.

unsigned short *Aseba*::*BytecodeVector*`getTypeOfLast`() `const`
Return the type of last bytecode element.

BytecodeVector::*EventAddressesToIdsMap* *Aseba*::*BytecodeVector***getEventAddressesToIds**() **const**

Get the map of event addresses to identifiers.

Public Members

unsigned *Aseba*::*BytecodeVector***maxStackDepth** = {0}

maximum depth of the stack used by all the computations of the bytecode

unsigned *Aseba*::*BytecodeVector***callDepth** = {0}

for callable bytecode (i.e.

subroutines), used in analysis of stack check

unsigned *Aseba*::*BytecodeVector***lastLine** = {0}

last line added, normally equal *this[this->size()-1].line, but may differ for instance on loops

struct *Aseba***CallNode** : **public** *Aseba*::*Node*

#include <tree.h> *Node* for calling a native function may have children for pushing constants somewhere.

Public Functions

Aseba::*CallNode***CallNode**(**const** *SourcePos* &*sourcePos*, unsigned *funcId*)

Constructor.

CallNode **Aseba*::*CallNode***shallowCopy**() **const**

Return a shallow copy of the object (children point to the same objects)

ReturnType *Aseba*::*CallNode***typeCheck**(*Compiler* **compiler*)

Typecheck this node, throw an exception if there is any type violation.

Node **Aseba*::*CallNode***optimize**(std::wostream **dump*)

Optimize this node, return the optimized node.

unsigned *Aseba*::*CallNode***getStackDepth**() **const**

Return the stack depth requirement for this node and its children.

void *Aseba*::*CallNode***emit**(*PreLinkBytecode* &*bytecodes*) **const**

Generate bytecode.

std::wstring *Aseba*::*CallNode***toWString**() **const**

Return a string representation of this node.

std::wstring *Aseba*::*CallNode***toNodeName**() **const**

Return a string representation of the name of this node.

Public Members

unsigned *Aseba*::*CallNode***funcId**

identifier of the function to be called

std::vector<unsigned> *Aseba*::*CallNode***templateArgs**

sizes of templated arguments

struct *Aseba***CallSubNode** : **public** *Aseba*::*Node*

#include <tree.h> *Node* for L"callsub" no children.

Public Functions

Aseba::*CallSubNode*`CallSubNode`(`const` *SourcePos* &*sourcePos*, std::wstring *subroutineName*)
Constructor.

CallSubNode **Aseba*::*CallSubNode*`shallowCopy`() `const`
Return a shallow copy of the object (children point to the same objects)

Node::*ReturnType Aseba*::*CallSubNode*`typeCheck`(*Compiler* **compiler*)
Typecheck this node, throw an exception if there is any type violation.

Node **Aseba*::*CallSubNode*`optimize`(std::wostream **dump*)
Optimize this node, return the optimized node.

void *Aseba*::*CallSubNode*`emit`(*PreLinkBytecode* &*bytecodes*) `const`
Generate bytecode.

std::wstring *Aseba*::*CallSubNode*`toWString`() `const`
Return a string representation of this node.

std::wstring *Aseba*::*CallSubNode*`toNodeName`() `const`
Return a string representation of the name of this node.

Public Members

std::wstring *Aseba*::*CallSubNode*`subroutineName`
the subroutine to call

unsigned *Aseba*::*CallSubNode*`subroutineId`

`struct` *Aseba*`CommonDefinitions`
#include <*compiler.h*> Definitions common to several nodes, such as events or some constants.

Public Functions

void *Aseba*::*CommonDefinitions*`clear`()
Clear all the content.

Public Members

EventsDescriptionsVector *Aseba*::*CommonDefinitions*`events`
All defined events.

ConstantsDefinitions *Aseba*::*CommonDefinitions*`constants`
All globally defined constants.

`class` *Aseba*`Compiler`
#include <*compiler.h*> Aseba Event Scripting Language compiler.

Public Types

using *Aseba*::*Compiler***SubroutineTable** = std::vector<*SubroutineDescriptor*>
Lookup table for subroutines id => (name, address, line)

typedef std::map<std::wstring, unsigned> *Aseba*::*Compiler***SubroutineReverseTable**
Reverse Lookup table for subroutines name => id.

using *Aseba*::*Compiler***ImplementedEvents** = std::set<unsigned int>
Lookup table to keep track of implemented events.

typedef std::map<std::wstring, int> *Aseba*::*Compiler***ConstantsMap**
Lookup table for constant name => value.

typedef std::map<std::wstring, unsigned> *Aseba*::*Compiler***EventsMap**
Lookup table for event name => id.

Public Functions

Aseba::*Compiler***Compiler**()
Constructor.

You must setup a description using *setTargetDescription()* before any call to *compile()*.

void *Aseba*::*Compiler***setTargetDescription**(**const** TargetDescription **description*)
Set the description of the target as returned by the microcontroller.

You must call this function before any call to *compile()*.

const TargetDescription **Aseba*::*Compiler***getTargetDescription**() **const**

const VariablesMap **Aseba*::*Compiler***getVariablesMap**() **const**

const *SubroutineTable* **Aseba*::*Compiler***getSubroutineTable**() **const**

void *Aseba*::*Compiler***setCommonDefinitions**(**const** *CommonDefinitions* **definitions*)
Set the common definitions, such as events or some constants.

bool *Aseba*::*Compiler***compile**(std::wistream *&source*, *BytecodeVector &bytecode*, unsigned *&allocatedVariablesCount*, *Error &errorDescription*, std::wostream **dump* = nullptr)
Compile a new condition.

Return returns true on success

Parameters

- `source`: stream to read the source code from
- `bytecode`: destination array for bytecode
- `allocatedVariablesCount`: amount of allocated variables
- `errorDescription`: error is copied there on error
- `dump`: stream to send dump messages to

void *Aseba*::*Compiler***setTranslateCallback**(*ErrorMessages*::*ErrorCallback newCB*)

Public Static Functions

static std::wstring *Aseba*::*Compiler***translate**(*ErrorCode error*)

bool *Aseba*::*Compiler***isKeyword**(**const** std::wstring &*word*)
Return whether a string is a language keyword.

Protected Functions

void *Aseba*::*Compiler***internalCompilerError**() **const**
There is a bug in the compiler, ask for a bug report.

void *Aseba*::*Compiler***expect**(**const** *Token*::*Type* &*type*) **const**
Check if next token is of type, produce an exception otherwise.

unsigned *Aseba*::*Compiler***expectUInt12Literal**() **const**
Check if next token is an unsigned 12 bits integer literal.

If so, return it, if not, throw an exception

unsigned *Aseba*::*Compiler***expectUInt16Literal**() **const**
Check if next token is an unsigned 16 bits integer literal.

If so, return it, if not, throw an exception

unsigned *Aseba*::*Compiler***expectPositiveInt16Literal**() **const**
Check if next token is the positive part of a 16 bits signed integer literal.

If so, return it, if not, throw an exception

int *Aseba*::*Compiler***expectAbsoluteInt16Literal**(bool *negative*) **const**
Check if next token is the absolute part of a 16 bits signed integer literal.

If so, return it, if not, throw an exception

unsigned *Aseba*::*Compiler***expectPositiveConstant**() **const**
Check if next toxen is a valid positive part of a 16 bits signed integer constant.

int *Aseba*::*Compiler***expectConstant**() **const**
Check if next toxen is a valid 16 bits signed integer constant.

unsigned *Aseba*::*Compiler***expectPositiveInt16LiteralOrConstant**() **const**
Check and return either the positive part of a 16 bits signed integer or the value of a valid constant.

int *Aseba*::*Compiler***expectInt16Literal**()
Check and return a 16 bits signed integer.

int *Aseba*::*Compiler***expectInt16LiteralOrConstant**()
Check and return either a 16 bits signed integer or the value of a valid constant.

unsigned *Aseba*::*Compiler***expectGlobalEventId**() **const**
Check if next token is a known global event identifier.

unsigned *Aseba*::*Compiler***expectAnyEventId**() **const**
Check if next token is a known local or global event identifier.

std::wstring *Aseba*::*Compiler***eventName**(unsigned *eventId*) **const**
Return the name of an event given its identifier.

template <int *length*>
bool *Aseba*::*Compiler***isOneOf**(**const** *Token*::*Type types*[*length*]) **const**
Return true if next token is of the following types.

template <int *length*>
void *Aseba*::*Compiler***expectOneOf**(**const** *Token*::*Type types*[*length*]) **const**
Check if next token is of one of the following types, produce an exception otherwise.

void *Aseba*::*Compiler***freeTemporaryMemory**()

unsigned *Aseba*::*Compiler***allocateTemporaryMemory**(**const** *SourcePos varPos*, **const** unsigned *size*)

AssignmentNode **Aseba*::*Compiler***allocateTemporaryVariable**(**const** *SourcePos varPos*, *Node* **rValue*)

VariablesMap::const_iterator *Aseba*::*Compiler***findVariable**(**const** std::wstring &*name*, **const** *SourcePos* &*pos*) **const**
Look for a variable of a given name, and if found, return an iterator; if not, return an exception.

FunctionsMap::const_iterator *Aseba*::*Compiler***findFunction**(**const** std::wstring &*name*, **const** *SourcePos* &*pos*) **const**
Look for a function of a given name, and if found, return an iterator; if not, return an exception.

Compiler::*ConstantsMap*::const_iterator *Aseba*::*Compiler***findConstant**(**const** std::wstring &*name*, **const** *SourcePos* &*pos*) **const**
Look for a constant of a given name, and if found, return an iterator; if not, return an exception.

Compiler::*EventsMap*::const_iterator *Aseba*::*Compiler***findGlobalEvent**(**const** std::wstring &*name*, **const** *SourcePos* &*pos*) **const**
Look for a global event of a given name, and if found, return an iterator; if not, return an exception.

Compiler::*EventsMap*::const_iterator *Aseba*::*Compiler***findAnyEvent**(**const** std::wstring &*name*, **const** *SourcePos* &*pos*) **const**

Compiler::*SubroutineReverseTable*::const_iterator *Aseba*::*Compiler***findSubroutine**(**const** std::wstring &*name*, **const** *SourcePos* &*pos*) **const**
Look for a subroutine of a given name, and if found, return an iterator; if not, return an exception.

bool *Aseba*::*Compiler***constantExists**(**const** std::wstring &*name*) **const**
Return true if a constant of a given name exists.

void *Aseba*::*Compiler***buildMaps**()
Build variables and functions maps.

void *Aseba*::*Compiler***tokenize**(std::wistream &*source*)
Parse source and build tokens vector.

Parameters

- `source`: source code

wchar_t *Aseba*::*Compiler***getNextCharacter**(std::wistream &*source*, *SourcePos* &*pos*)

bool *Aseba*::*Compiler*`testNextCharacter`(std::wistream &*source*, *SourcePos* &*pos*, wchar_t *test*, *Token*::*Type* *tokenIfTrue*)

void *Aseba*::*Compiler*`dumpTokens`(std::wostream &*dest*) `const`
Debug print of tokens.

bool *Aseba*::*Compiler*`verifyStackCalls`(*PreLinkBytecode* &*preLinkBytecode*)
Verify that no call path can create a stack overflow.

bool *Aseba*::*Compiler*`link`(`const` *PreLinkBytecode* &*preLinkBytecode*, *BytecodeVector* &*bytecode*)
Create the final bytecode for a microcontroller.

void *Aseba*::*Compiler*`disassemble`(*BytecodeVector* &*bytecode*, `const` *PreLinkBytecode* &*preLinkBytecode*, std::wostream &*dump*) `const`
Disassemble a microcontroller bytecode and dump it.

Node **Aseba*::*Compiler*`parseProgram`()
Parse "program" grammar element.

Node **Aseba*::*Compiler*`parseStatement`()
Parse "statement" grammar element.

Node **Aseba*::*Compiler*`parseBlockStatement`()
Parse "block statement" grammar element.

Node **Aseba*::*Compiler*`parseReturn`()
Parse "return statement" grammar element.

void *Aseba*::*Compiler*`parseConstDef`()
Parse "const def" elements.

Node **Aseba*::*Compiler*`parseVarDef`()
Parse "var def" grammar element.

AssignmentNode **Aseba*::*Compiler*`parseVarDefInit`(*MemoryVectorNode* **lValue*)

Node **Aseba*::*Compiler*`parseAssignment`()
Parse "assignment" grammar element.

Node **Aseba*::*Compiler*`parseIfWhen`(bool *edgeSensitive*)
Parse "if" grammar element.

Node **Aseba*::*Compiler*`parseFor`()
Parse "for" grammar element.

Node **Aseba*::*Compiler*`parseWhile`()
Parse "while" grammar element.

Node **Aseba*::*Compiler*`parseOnEvent`()
Parse "onevent" grammar element.

Node **Aseba*::*Compiler*`parseEmit`(bool *shorterArgsAllowed* = false)
Parse "event" grammar element.

Node **Aseba*::*Compiler*`parseSubDecl`()
Parse "sub" grammar element, declaration of subroutine.

Node **Aseba*::*Compiler*`parseCallSub`()
Parse "subcall" grammar element, call of subroutine.

Node **Aseba*::*Compiler*::**parseOr**()
Parse "or" grammar element.

Node **Aseba*::*Compiler*::**parseAnd**()
Parse "and" grammar element.

Node **Aseba*::*Compiler*::**parseNot**()
Parse "not" grammar element.

Node **Aseba*::*Compiler*::**parseCondition**()
Parse "condition" grammar element.

Node **Aseba*::*Compiler*::**parseBinaryOrExpression**()
Parse "binary or" grammar element.

Node **Aseba*::*Compiler*::**parseBinaryXorExpression**()
Parse "binary xor" grammar element.

Node **Aseba*::*Compiler*::**parseBinaryAndExpression**()
Parse "binary and" grammar element.

Node **Aseba*::*Compiler*::**parseShiftExpression**()
Parse "shift_expression" grammar element.

Node **Aseba*::*Compiler*::**parseAddExpression**()
Parse "add_expression" grammar element.

Node **Aseba*::*Compiler*::**parseMultExpression**()
Parse "mult_expression" grammar element.

Node **Aseba*::*Compiler*::**parseUnaryExpression**()
Parse "unary_expression" grammar element.

Node **Aseba*::*Compiler*::**parseFunctionCall**()
Parse "function_call" grammar element.

TupleVectorNode **Aseba*::*Compiler*::**parseTupleVector**(bool *compatibility* = false)
Parse "[....]" grammar element.

Node **Aseba*::*Compiler*::**parseConstantAndVariable**()

MemoryVectorNode **Aseba*::*Compiler*::**parseVariable**()

unsigned *Aseba*::*Compiler*::**parseVariableDefSize**()

Node **Aseba*::*Compiler*::**tryParsingConstantExpression**(*SourcePos* *pos*, int &*constantResult*)
Use *parseBinaryOrExpression()* and try to reduce the result into an integer If successful, return nullptr and the integer in constantResult If unsuccessful, return the parsed tree (constantResult useless in this case)

int *Aseba*::*Compiler*::**expectConstantExpression**(*SourcePos* *pos*, *Node* **tree*)
This is a generalization of *expectPositiveInt16LiteralOrConstant()* Try to reduce the expression into a single figure, if not raise an exception The tree pointed by "tree" is deleted during execution, not safe to use it after.

Protected Attributes

std::deque<*Token*> *Aseba*::*Compiler***tokens**
parsed tokens

VariablesMap *Aseba*::*Compiler***variablesMap**
variables lookup

ImplementedEvents Aseba::*Compiler***implementedEvents**
list of implemented events

FunctionsMap *Aseba*::*Compiler***functionsMap**
functions lookup

ConstantsMap Aseba::*Compiler***constantsMap**
constants map

EventsMap Aseba::*Compiler***globalEventsMap**
global-events map

EventsMap Aseba::*Compiler***allEventsMap**
all-events map

SubroutineTable Aseba::*Compiler***subroutineTable**
subroutine lookup

SubroutineReverseTable Aseba::*Compiler***subroutineReverseTable**
subroutine reverse lookup

unsigned *Aseba*::*Compiler***freeVariableIndex**
index pointing to the first free variable

unsigned *Aseba*::*Compiler***endVariableIndex**
(endMemory - endVariableIndex) is pointing to the first free variable at the end

const TargetDescription **Aseba*::*Compiler***targetDescription**
description of the target VM

const *CommonDefinitions* **Aseba*::*Compiler***commonDefinitions**
common definitions, such as events or some constants

ErrorMessages Aseba::*Compiler***translator**

Friends

friend Aseba::Compiler::AssignmentNode

friend Aseba::Compiler::CallSubNode

struct *Aseba***EmitNode** : **public** *Aseba*::*Node*
#include <tree.h> Node for L"emit".

may have children for pushing constants somewhere

Public Functions

Aseba::*EmitNode***EmitNode**(**const** *SourcePos* &*sourcePos*)
Constructor.

EmitNode **shallowCopy**() **const*
Return a shallow copy of the object (children point to the same objects)

ReturnType **typeCheck**(*Compiler* **compiler*)
Typecheck this node, throw an exception if there is any type violation.

Node **optimize**(std::wostream **dump*)
Optimize this node, return the optimized node.

void **emit**(*PreLinkBytecode* &*bytecodes*) **const**
Generate bytecode.

std::wstring **toWString**() **const**
Return a string representation of this node.

std::wstring **toNodeName**() **const**
Return a string representation of the name of this node.

Public Members

unsigned **eventId**
id of event to emit

unsigned **arrayAddr**
address of the first element of the array to send

unsigned **arraySize**
size of the array to send.

0 if event has no argument

struct *Aseba***Error**
#include <*compiler.h*> Compilation error.

Subclassed by *Aseba::TranslatableError*

Public Functions

Error(**const** *SourcePos* &*pos*, std::wstring *message*)
Create an error at pos.

Error()
Create an empty error.

std::wstring **toWString**() **const**
Return a string describing the error.

Return the string version of this error.

Public Members

SourcePos **pos**
position of error in source string

std::wstring **message** = L"not defined"
message

struct *Aseba***ErrorMessages**

#include <compiler.h> Return error messages based on error ID (needed to translate messages for other applications)

Public Types

using *Aseba*::*ErrorMessages***ErrorCallback** = **const** std::wstring (*) (*ErrorCode*)

Type of the callback.

Public Functions

Aseba::*ErrorMessages***ErrorMessages**()

Public Static Functions

const std::wstring *Aseba*::*ErrorMessages***defaultCallback**(*ErrorCode error*)

Default callback.

struct *Aseba***EventDeclNode** : **public** *Aseba*::*Node*

#include <tree.h> *Node* for L"onevent" no children.

Public Functions

Aseba::*EventDeclNode***EventDeclNode**(**const** *SourcePos* &*sourcePos*, unsigned *eventId* = 0)

Constructor.

EventDeclNode **Aseba*::*EventDeclNode***shallowCopy**() **const**

Return a shallow copy of the object (children point to the same objects)

ReturnType *Aseba*::*EventDeclNode***typeCheck**(*Compiler* **compiler*)

Typecheck this node, throw an exception if there is any type violation.

Node **Aseba*::*EventDeclNode***optimize**(std::wostream **dump*)

Optimize this node, return the optimized node.

void *Aseba*::*EventDeclNode***emit**(*PreLinkBytecode* &*bytecodes*) **const**

Generate bytecode.

std::wstring *Aseba*::*EventDeclNode***toWString**() **const**

Return a string representation of this node.

std::wstring *Aseba*::*EventDeclNode***toNodeName**() **const**

Return a string representation of the name of this node.

Public Members

unsigned *Aseba*::*EventDeclNode***eventId**

the event id associated with this context

struct *Aseba***FoldedIfWhenNode** : **public** *Aseba*::*Node*
#include <tree.h> *Node* for L"if" and L"when" with operator folded inside.

children[0] is left part of expression children[1] is right part of expression children[2] is true block children[3] is false block

Public Functions

Aseba::*FoldedIfWhenNode***FoldedIfWhenNode**(**const** *SourcePos* &*sourcePos*)
Constructor.

FoldedIfWhenNode **Aseba*::*FoldedIfWhenNode***shallowCopy**() **const**
Return a shallow copy of the object (children point to the same objects)

void *Aseba*::*FoldedIfWhenNode***checkVectorSize**() **const**
Check the consistency in vectors' size.

Node **Aseba*::*FoldedIfWhenNode***optimize**(std::wostream **dump*)
Optimize this node, return the optimized node.

unsigned *Aseba*::*FoldedIfWhenNode***getStackDepth**() **const**
Return the stack depth requirement for this node and its children.

void *Aseba*::*FoldedIfWhenNode***emit**(*PreLinkBytecode* &*bytecodes*) **const**
Generate bytecode.

std::wstring *Aseba*::*FoldedIfWhenNode***toWString**() **const**
Return a string representation of this node.

std::wstring *Aseba*::*FoldedIfWhenNode***toNodeName**() **const**
Return a string representation of the name of this node.

Public Members

AsebaBinaryOperator *Aseba*::*FoldedIfWhenNode***op**
operator

bool *Aseba*::*FoldedIfWhenNode***edgeSensitive**
if true, true block is triggered only if previous comparison was false ("when" block).

If false, true block is triggered every time comparison is true

unsigned *Aseba*::*FoldedIfWhenNode***endLine**
line of end keyword

struct *Aseba***FoldedWhileNode** : **public** *Aseba*::*Node*
#include <tree.h> *Node* for L"while" with operator folded inside.

children[0] is left part of expression children[1] is right part of expression children[2] is block

Public Functions

Aseba::*FoldedWhileNode***FoldedWhileNode**(**const** *SourcePos* &*sourcePos*)
Constructor.

FoldedWhileNode *Aseba*::*FoldedWhileNode***shallowCopy**() **const**
Return a shallow copy of the object (children point to the same objects)

void *Aseba*::*FoldedWhileNode***checkVectorSize**() **const**
Check the consistency in vectors' size.

Node **Aseba*::*FoldedWhileNode***optimize**(std::wostream **dump*)
Optimize this node, return the optimized node.

unsigned *Aseba*::*FoldedWhileNode***getStackDepth**() **const**
Return the stack depth requirement for this node and its children.

void *Aseba*::*FoldedWhileNode***emit**(*PreLinkBytecode* &*bytecodes*) **const**
Generate bytecode.

std::wstring *Aseba*::*FoldedWhileNode***toWString**() **const**
Return a string representation of this node.

std::wstring *Aseba*::*FoldedWhileNode***toNodeName**() **const**
Return a string representation of the name of this node.

Public Members

AsebaBinaryOperator *Aseba*::*FoldedWhileNode***op**
operator

struct *Aseba***IfWhenNode** : **public** *Aseba*::*Node*
#include <tree.h> *Node* for L"if" and L"when".

children[0] is expression children[1] is true block children[2] is false block

Public Functions

Aseba::*IfWhenNode***IfWhenNode**(**const** *SourcePos* &*sourcePos*)
Constructor.

IfWhenNode **Aseba*::*IfWhenNode***shallowCopy**() **const**
Return a shallow copy of the object (children point to the same objects)

void *Aseba*::*IfWhenNode***checkVectorSize**() **const**
Check the consistency in vectors' size.

Node::*ReturnType Aseba*::*IfWhenNode***typeCheck**(*Compiler* **compiler*)
Typecheck this node, throw an exception if there is any type violation.

Node **Aseba*::*IfWhenNode***optimize**(std::wostream **dump*)
Optimize this node, return the optimized node.

void *Aseba*::*IfWhenNode***emit**(*PreLinkBytecode* &*bytecodes*) **const**
Generate bytecode.

std::wstring *Aseba*::*IfWhenNode***toWString**() **const**
Return a string representation of this node.

std::wstring *Aseba*::*IfWhenNode***toNodeName**() **const**
Return a string representation of the name of this node.

Public Members

bool *Aseba*::*IfWhenNode*`edgeSensitive`
if true, true block is triggered only if previous comparison was false ("when" block).

If false, true block is triggered every time comparison is true

unsigned *Aseba*::*IfWhenNode*`endLine`
line of end keyword

`struct` *Aseba*`ImmediateNode` : `public` *Aseba*::*Node*
#include <tree.h> *Node* for pushing immediate value on stack.

Might generate either Small Immediate or Large Immediate bytecode depending on the range of value no children

Public Functions

Aseba::*ImmediateNode*`ImmediateNode`(`const` *SourcePos* &*sourcePos*, int *value*)
Constructor.

ImmediateNode **Aseba*::*ImmediateNode*`shallowCopy`() `const`
Return a shallow copy of the object (children point to the same objects)

Node **Aseba*::*ImmediateNode*`expandVectorialNodes`(std::wostream **dump*, *Compiler* **compiler* = nullptr, unsigned int *index* = 0)
Expand into a copy of itself.

ReturnType *Aseba*::*ImmediateNode*`typeCheck`(*Compiler* **compiler*)
Typecheck this node, throw an exception if there is any type violation.

Node **Aseba*::*ImmediateNode*`optimize`(std::wostream **dump*)
Optimize this node, return the optimized node.

unsigned *Aseba*::*ImmediateNode*`getStackDepth`() `const`
Return the stack depth requirement for this node and its children.

void *Aseba*::*ImmediateNode*`emit`(*PreLinkBytecode* &*bytecodes*) `const`
Generate bytecode.

std::wstring *Aseba*::*ImmediateNode*`toWString`() `const`
Return a string representation of this node.

std::wstring *Aseba*::*ImmediateNode*`toNodeName`() `const`
Return a string representation of the name of this node.

unsigned *Aseba*::*ImmediateNode*`getVectorSize`() `const`
return the children's size, check for equal size, or E_NOVAL if no child

Public Members

int *Aseba*::*ImmediateNode*`value`
value to push on stack

`struct` *Aseba*`LoadNativeArgNode` : `public` *Aseba*::*Node*
#include <tree.h> *Node* for loading the address of the argument of a native function that is not known at compile time, but that does exist in memory.

Public Functions

Aseba::*LoadNativeArgNode***LoadNativeArgNode**(*MemoryVectorNode* **memoryNode*, unsigned *tempAddr*)
Constructor, delete the provided memoryNode.

LoadNativeArgNode **Aseba*::*LoadNativeArgNode***shallowCopy**() **const**
Return a shallow copy of the object (children point to the same objects)

ReturnType *Aseba*::*LoadNativeArgNode***typeCheck**(*Compiler* **compiler*)
Typecheck this node, throw an exception if there is any type violation.

Node **Aseba*::*LoadNativeArgNode***optimize**(std::wostream **dump*)
Optimize this node, return the optimized node.

unsigned *Aseba*::*LoadNativeArgNode***getStackDepth**() **const**
Return the stack depth requirement for this node and its children.

void *Aseba*::*LoadNativeArgNode***emit**(*PreLinkBytecode* &*bytecodes*) **const**
Generate bytecode.

std::wstring *Aseba*::*LoadNativeArgNode***toWString**() **const**
Return a string representation of this node.

std::wstring *Aseba*::*LoadNativeArgNode***toNodeName**() **const**
Return a string representation of the name of this node.

Public Members

unsigned *Aseba*::*LoadNativeArgNode***tempAddr**
address of temporary (end of memory) for stack value duplication

unsigned *Aseba*::*LoadNativeArgNode***arrayAddr**
address of the first element of the array

unsigned *Aseba*::*LoadNativeArgNode***arraySize**
size of the array, might be used to assert compile-time access checks

std::wstring *Aseba*::*LoadNativeArgNode***arrayName**
name of the array (for debug)

struct *Aseba***LoadNode** : **public** *Aseba*::*Node*
#include <tree.h> *Node* for loading a variable on stack.

no children

Public Functions

Aseba::*LoadNode***LoadNode**(**const** *SourcePos* &*sourcePos*, unsigned *varAddr*)
Constructor.

LoadNode **Aseba*::*LoadNode***shallowCopy**() **const**
Return a shallow copy of the object (children point to the same objects)

ReturnType *Aseba*::*LoadNode***typeCheck**(*Compiler* **compiler*)
Typecheck this node, throw an exception if there is any type violation.

Node *Aseba*::*LoadNode***optimize**(std::wostream *dump)
Optimize this node, return the optimized node.

unsigned *Aseba*::*LoadNode***getStackDepth**() **const**
Return the stack depth requirement for this node and its children.

void *Aseba*::*LoadNode***emit**(*PreLinkBytecode* &bytecodes) **const**
Generate bytecode.

std::wstring *Aseba*::*LoadNode***toWString**() **const**
Return a string representation of this node.

std::wstring *Aseba*::*LoadNode***toNodeName**() **const**
Return a string representation of the name of this node.

unsigned *Aseba*::*LoadNode***getVectorAddr**() **const**
return the address of the left-most operand, or E_NOVAL if none

unsigned *Aseba*::*LoadNode***getVectorSize**() **const**
return the children's size, check for equal size, or E_NOVAL if no child

Public Members

unsigned *Aseba*::*LoadNode***varAddr**
address of variable to load from

struct *Aseba***MemoryVectorNode** : **public** *Aseba*::*AbstractTreeNode*
#include <tree.h> *Node* for accessing a memory as a vector, in read or write operations If write == true, will expand to *StoreNode* or *ArrayWriteNode* If write == false, will expand to *LoadNode* or *ArrayReadNode* children[0] is an optional index If children[0] is a *TupleVectorNode* of one elements (int), it will be foo[x] If children[0] is a *TupleVectorNode* of two elements (int), it will be foo[x:y] If children[0] is another type of node, it will be foo[whatever] If children[0] doesn't exist, access to the full array is considered.

Public Functions

Aseba::*MemoryVectorNode***MemoryVectorNode**(**const** *SourcePos* &*sourcePos*, unsigned *arrayAddr*, unsigned *arraySize*, std::wstring *arrayName*)
Constructor.

MemoryVectorNode *Aseba*::*MemoryVectorNode***shallowCopy**() **const**
Return a shallow copy of the object (children point to the same objects)

Node *Aseba*::*MemoryVectorNode***expandAbstractNodes**(std::wostream *dump)
Dummy function, the node will be expanded during the vectorial pass.

Node *Aseba*::*MemoryVectorNode***expandVectorialNodes**(std::wostream *dump, *Compiler* *compiler = nullptr, unsigned int *index* = 0)
Expand to memory[index].

std::wstring *Aseba*::*MemoryVectorNode***toWString**() **const**
Return a string representation of this node.

std::wstring *Aseba*::*MemoryVectorNode***toNodeName**() **const**
Return a string representation of the name of this node.

unsigned *Aseba*::*MemoryVectorNode*`getVectorAddr` () `const`
return the compile-time base address of the memory range, taking into account an immediate index foo[n] or foo[n:m] return E_NOVAL if foo[expr]

unsigned *Aseba*::*MemoryVectorNode*`getVectorSize` () `const`
return the vector's length

bool *Aseba*::*MemoryVectorNode*`isAddressStatic` () `const`
return whether this node accesses a static address

`virtual` void *Aseba*::*MemoryVectorNode*`setWrite` (bool *write*)

Public Members

unsigned *Aseba*::*MemoryVectorNode*`arrayAddr`
address of the first element of the array

unsigned *Aseba*::*MemoryVectorNode*`arraySize`
size of the array, might be used to assert compile-time access checks

std::wstring *Aseba*::*MemoryVectorNode*`arrayName`
name of the array (for debug)

bool *Aseba*::*MemoryVectorNode*`write`
expand to a node for storing or loading data?

`struct` *Aseba*`NamedValue`
#include <*compiler.h*> A name - value pair.

Public Functions

Aseba::*NamedValue*`NamedValue` (std::wstring *name*, int *value*)
Create a filled pair.

Public Members

std::wstring *Aseba*::*NamedValue*`name`
name part of the pair

int *Aseba*::*NamedValue*`value`
value part of the pair

`struct` *Aseba*`NamedValuesVector` : `public` std::vector<*NamedValue*>
#include <*compiler.h*> Generic vector of name - value pairs.

Public Functions

bool *Aseba*::*NamedValuesVector*`contains` (`const` std::wstring &*s*, size_t **position* = nullptr) `const`
Return whether a named-value vector contains a certain value, by iterating.

`struct` *Aseba*`Node`
#include <*tree.h*> An abstract node of syntax tree.

Subclassed by *Aseba::AbstractTreeNode*, *Aseba::ArrayReadNode*, *Aseba::ArrayWriteNode*, *Aseba::AssignmentNode*, *Aseba::BinaryArithmeticNode*, *Aseba::BlockNode*, *Aseba::CallNode*, *Aseba::CallSubNode*, *Aseba::EmitNode*, *Aseba::EventDeclNode*, *Aseba::FoldedIfWhenNode*, *Aseba::FoldedWhileNode*, *Aseba::IfWhenNode*, *Aseba::ImmediateNode*, *Aseba::LoadNativeArgNode*, *Aseba::LoadNode*, *Aseba::ReturnNode*, *Aseba::StoreNode*, *Aseba::SubDeclNode*, *Aseba::UnaryArithmeticNode*, *Aseba::WhileNode*

Public Types

enum *Aseba*::*Node***ReturnType**
A type a node can return.

Values:

Aseba::*Node***UNIT** = 0

Aseba::*Node***BOOL**

Aseba::*Node***INT**

enum *Aseba*::*Node***MemoryErrorCode**
Values:

Aseba::*Node***E_NOVAL** = UINT_MAX

using *Aseba*::*Node***NodesVector** = std::vector<*Node* *>
Vector for children of a node.

Public Functions

Aseba::*Node***Node**(**const** *SourcePos* &*sourcePos*)
Constructor.

Node &*Aseba*::*Node***operator=**(**const** *Node*&)

Aseba::*Node***Node**(*Node*&&)

Node &*Aseba*::*Node***operator=**(*Node* &&*rhs*)

Aseba::*Node***~Node**()
Destructor, delete all children.

virtual *Node* **Aseba*::*Node***shallowCopy**() **const** = 0
Return a shallow copy of the object (children point to the same objects)

Node **Aseba*::*Node***deepCopy**() **const**
Return a deep copy of the object (children are also copied)

void *Aseba*::*Node***checkVectorSize**() **const**
Check the consistency in vectors' size.

Node **Aseba*::*Node***expandAbstractNodes**(std::wostream **dump*)
Second pass to expand "abstract" nodes into more concrete ones.

Generically traverse the tree, expand the children, and perform garbagge collection.

Node *Aseba*::*Node***expandVectorialNodes**(std::wostream *dump, *Compiler* *compiler = nullptr, unsigned int *index* = 0)
Third pass to expand vectorial operations into mutliple scalar ones.

Generic implementation for non-vectorial nodes.

Node::*ReturnType* *Aseba*::*Node***typeCheck**(*Compiler* *compiler)
Typecheck this node, throw an exception if there is any type violation.

virtual *Node* *Aseba*::*Node***optimize**(std::wostream *dump) = 0
Optimize this node, return the optimized node.

unsigned *Aseba*::*Node***getStackDepth**() **const**
Return the stack depth requirement for this node and its children.

virtual void *Aseba*::*Node***emit**(*PreLinkBytecode* &*bytecodes*) **const** = 0
Generate bytecode.

virtual std::wstring *Aseba*::*Node***toWString**() **const** = 0
Return a string representation of this node.

virtual std::wstring *Aseba*::*Node***toNodeName**() **const** = 0
Return a string representation of the name of this node.

void *Aseba*::*Node***dump**(std::wostream &*dest*, unsigned &*indent*) **const**
Dump this node and the rest of the tree.

std::wstring *Aseba*::*Node***typeName**(**const** *Node*::*ReturnType* &*type*) **const**
Return the name of a type.

void *Aseba*::*Node***expectType**(**const** *Node*::*ReturnType* &*expected*, **const** *Node*::*ReturnType* &*type*) **const**
Check for a specific type, throw an exception otherwise.

unsigned *Aseba*::*Node***getVectorAddr**() **const**
return the address of the left-most operand, or E_NOVAL if none

unsigned *Aseba*::*Node***getVectorSize**() **const**
return the children's size, check for equal size, or E_NOVAL if no child

Public Members

NodesVector *Aseba*::*Node***children**
children of this node

SourcePos *Aseba*::*Node***sourcePos**
position is source

Protected Functions

Aseba::*Node***Node**(**const** *Node*&)

struct *Aseba***PreLinkBytecode**
#include <compiler.h> Bytecode use for compilation previous to linking.

Public Types

typedef std::map<unsigned, *BytecodeVector*> *Aseba*::*PreLinkBytecode***EventsBytecode**
Map of events id to event bytecode.

typedef std::map<unsigned, *BytecodeVector*> *Aseba*::*PreLinkBytecode***SubroutinesBytecode**
Map of routines id to routine bytecode.

Public Functions

Aseba::*PreLinkBytecode***PreLinkBytecode**()
Add init event and point to currentBytecode it.

void *Aseba*::*PreLinkBytecode***fixup**(**const** *Compiler*::*SubroutineTable* &*subroutineTable*)
Fixup prelinked bytecodes by making sure that each vector is closed correctly, i.e.

with a STOP for events and a RET for subroutines

Public Members

EventsBytecode Aseba::*PreLinkBytecode***events**
bytecode for events

SubroutinesBytecode Aseba::*PreLinkBytecode***subroutines**
bytecode for routines

BytecodeVector **Aseba*::*PreLinkBytecode***current**
pointer to bytecode being constructed

struct *Aseba***ProgramNode** : **public** *Aseba*::*BlockNode*
#include <tree.h> Node for L"program", i.e. a block node with some special behaviour later on.

Public Functions

Aseba::*ProgramNode***ProgramNode**(**const** *SourcePos* &*sourcePos*)
Constructor.

ProgramNode **Aseba*::*ProgramNode***shallowCopy**() **const**
Return a shallow copy of the object (children point to the same objects)

Node **Aseba*::*ProgramNode***expandVectorialNodes**(std::wostream **dump*, *Compiler* **compiler* = nullptr, unsigned int *index* = 0)
This is the root node, take in charge the tree creation / deletion.

void *Aseba*::*ProgramNode***emit**(*PreLinkBytecode* &*bytecodes*) **const**
Generate bytecode.

std::wstring *Aseba*::*ProgramNode***toWString**() **const**
Return a string representation of this node.

std::wstring *Aseba*::*ProgramNode***toNodeName**() **const**
Return a string representation of the name of this node.

struct *Aseba***ReturnNode** : **public** *Aseba*::*Node*
#include <tree.h> Node for returning from an event or subroutine has no children, just a jump of 0 offset that will be resolved at link time.

Public Functions

Aseba::*ReturnNode***ReturnNode**(**const** *SourcePos* &*sourcePos*)

ReturnNode **Aseba*::*ReturnNode***shallowCopy**() **const**
Return a shallow copy of the object (children point to the same objects)

ReturnType *Aseba*::*ReturnNode***typeCheck**(*Compiler* **compiler*)
Typecheck this node, throw an exception if there is any type violation.

Node **Aseba*::*ReturnNode***optimize**(std::wostream **dump*)
Optimize this node, return the optimized node.

unsigned *Aseba*::*ReturnNode***getStackDepth**() **const**
Return the stack depth requirement for this node and its children.

void *Aseba*::*ReturnNode***emit**(*PreLinkBytecode* &*bytecodes*) **const**
Generate bytecode.

std::wstring *Aseba*::*ReturnNode***toWString**() **const**
Return a string representation of this node.

std::wstring *Aseba*::*ReturnNode***toNodeName**() **const**
Return a string representation of the name of this node.

struct *Aseba***SourcePos**
#include <compiler.h> Position in a source file or string. First is line, second is column.

Public Functions

Aseba::*SourcePos***SourcePos**(unsigned *character*, unsigned *row*, unsigned *column*)

Aseba::*SourcePos***SourcePos**()

std::wstring *Aseba*::*SourcePos***toWString**() **const**
Return the string version of this position.

Public Members

unsigned *Aseba*::*SourcePos***character** = {0}
position in source text

unsigned *Aseba*::*SourcePos***row** = {0}
line in source text

unsigned *Aseba*::*SourcePos***column** = {0}
column in source text

bool *Aseba*::*SourcePos***valid** = {false}
true if character, row and column hold valid values

struct *Aseba***StoreNode** : **public** *Aseba*::*Node*
#include <tree.h> *Node* for storing a variable from stack.

no children

Public Functions

Aseba::*StoreNode***StoreNode**(**const** *SourcePos* &*sourcePos*, unsigned *varAddr*)
Constructor.

StoreNode **Aseba*::*StoreNode***shallowCopy**() **const**
Return a shallow copy of the object (children point to the same objects)

ReturnType *Aseba*::*StoreNode***typeCheck**(*Compiler* **compiler*)
Typecheck this node, throw an exception if there is any type violation.

Node **Aseba*::*StoreNode***optimize**(std::wostream **dump*)
Optimize this node, return the optimized node.

void *Aseba*::*StoreNode***emit**(*PreLinkBytecode* &*bytecodes*) **const**
Generate bytecode.

std::wstring *Aseba*::*StoreNode***toWString**() **const**
Return a string representation of this node.

std::wstring *Aseba*::*StoreNode***toNodeName**() **const**
Return a string representation of the name of this node.

unsigned *Aseba*::*StoreNode***getVectorAddr**() **const**
return the address of the left-most operand, or E_NOVAL if none

unsigned *Aseba*::*StoreNode***getVectorSize**() **const**
return the children's size, check for equal size, or E_NOVAL if no child

Public Members

unsigned *Aseba*::*StoreNode***varAddr**
address of variable to store to

struct *Aseba***SubDeclNode** : **public** *Aseba*::*Node*
#include <tree.h> *Node* for L"sub" no children.

Public Functions

Aseba::*SubDeclNode***SubDeclNode**(**const** *SourcePos* &*sourcePos*, unsigned *subroutineId*)
Constructor.

SubDeclNode **Aseba*::*SubDeclNode***shallowCopy**() **const**
Return a shallow copy of the object (children point to the same objects)

ReturnType *Aseba*::*SubDeclNode***typeCheck**(*Compiler* **compiler*)
Typecheck this node, throw an exception if there is any type violation.

Node **Aseba*::*SubDeclNode***optimize**(std::wostream **dump*)
Optimize this node, return the optimized node.

void *Aseba*::*SubDeclNode***emit**(*PreLinkBytecode* &*bytecodes*) **const**
Generate bytecode.

std::wstring *Aseba*::*SubDeclNode*`toWString() const`
Return a string representation of this node.

std::wstring *Aseba*::*SubDeclNode*`toNodeName() const`
Return a string representation of the name of this node.

Public Members

unsigned *Aseba*::*SubDeclNode*`subroutineId`
the associated subroutine

`struct` *Aseba*::*Compiler*`SubroutineDescriptor`
#include <*compiler.h*> Description of a subroutine.

Public Functions

Aseba::*Compiler*::*SubroutineDescriptor*`SubroutineDescriptor`(std::wstring *name*, unsigned *address*, unsigned *line*)

Public Members

std::wstring *Aseba*::*Compiler*::*SubroutineDescriptor*`name`

unsigned *Aseba*::*Compiler*::*SubroutineDescriptor*`address`

unsigned *Aseba*::*Compiler*::*SubroutineDescriptor*`line`

`struct` *Aseba*::*Compiler*`Token`
#include <*compiler.h*> A token is a parsed element of inputs.

Public Types

`enum` *Aseba*::*Compiler*::*Token*`Type`
Values:

Aseba::*Compiler*::*Token*`TOKEN_END_OF_STREAM` = 0

Aseba::*Compiler*::*Token*`TOKEN_STR_when`

Aseba::*Compiler*::*Token*`TOKEN_STR_emit`

Aseba::*Compiler*::*Token*`TOKEN_STR_hidden_emit`

Aseba::*Compiler*::*Token*`TOKEN_STR_for`

Aseba::*Compiler*::*Token*`TOKEN_STR_in`

Aseba::*Compiler*::*Token*`TOKEN_STR_step`

Aseba::*Compiler*::*Token*`TOKEN_STR_while`

Aseba::*Compiler*::*Token*`TOKEN_STR_do`

Aseba::*Compiler*::*Token*`TOKEN_STR_if`

Aseba::*Compiler*::*Token*`TOKEN_STR_then`

Aseba::*Compiler*::*Token*`TOKEN_STR_else`

Aseba::*Compiler*::*Token***TOKEN_STR_elseif**

Aseba::*Compiler*::*Token***TOKEN_STR_end**

Aseba::*Compiler*::*Token***TOKEN_STR_var**

Aseba::*Compiler*::*Token***TOKEN_STR_const**

Aseba::*Compiler*::*Token***TOKEN_STR_call**

Aseba::*Compiler*::*Token***TOKEN_STR_sub**

Aseba::*Compiler*::*Token***TOKEN_STR_callsub**

Aseba::*Compiler*::*Token***TOKEN_STR_onevent**

Aseba::*Compiler*::*Token***TOKEN_STR_abs**

Aseba::*Compiler*::*Token***TOKEN_STR_return**

Aseba::*Compiler*::*Token***TOKEN_STRING_LITERAL**

Aseba::*Compiler*::*Token***TOKEN_INT_LITERAL**

Aseba::*Compiler*::*Token***TOKEN_PAR_OPEN**

Aseba::*Compiler*::*Token***TOKEN_PAR_CLOSE**

Aseba::*Compiler*::*Token***TOKEN_BRACKET_OPEN**

Aseba::*Compiler*::*Token***TOKEN_BRACKET_CLOSE**

Aseba::*Compiler*::*Token***TOKEN_COLON**

Aseba::*Compiler*::*Token***TOKEN_COMMA**

Aseba::*Compiler*::*Token***TOKEN_ASSIGN**

Aseba::*Compiler*::*Token***TOKEN_OP_OR**

Aseba::*Compiler*::*Token***TOKEN_OP_AND**

Aseba::*Compiler*::*Token***TOKEN_OP_NOT**

Aseba::*Compiler*::*Token***TOKEN_OP_EQUAL**

Aseba::*Compiler*::*Token***TOKEN_OP_NOT_EQUAL**

Aseba::*Compiler*::*Token***TOKEN_OP_BIGGER**

Aseba::*Compiler*::*Token***TOKEN_OP_BIGGER_EQUAL**

Aseba::*Compiler*::*Token***TOKEN_OP_SMALLER**

Aseba::*Compiler*::*Token***TOKEN_OP_SMALLER_EQUAL**

Aseba::*Compiler*::*Token***TOKEN_OP_ADD**

Aseba::*Compiler*::*Token***TOKEN_OP_NEG**

Aseba::*Compiler*::*Token***TOKEN_OP_MULT**

Aseba::*Compiler*::*Token***TOKEN_OP_DIV**

Aseba::*Compiler*::*Token***TOKEN_OP_MOD**

Aseba::*Compiler*::*Token***TOKEN_OP_SHIFT_LEFT**

Aseba::*Compiler*::*Token***TOKEN_OP_SHIFT_RIGHT**

Aseba::*Compiler*::*Token***TOKEN_OP_BIT_OR**

Aseba::*Compiler*::*Token***TOKEN_OP_BIT_XOR**

Aseba::*Compiler*::*Token***TOKEN_OP_BIT_AND**

Aseba::*Compiler*::*Token***TOKEN_OP_BIT_NOT**

Aseba::*Compiler*::*Token***TOKEN_OP_ADD_EQUAL**

Aseba::*Compiler*::*Token***TOKEN_OP_NEG_EQUAL**

Aseba::*Compiler*::*Token***TOKEN_OP_MULT_EQUAL**

Aseba::*Compiler*::*Token***TOKEN_OP_DIV_EQUAL**

Aseba::*Compiler*::*Token***TOKEN_OP_MOD_EQUAL**

Aseba::*Compiler*::*Token***TOKEN_OP_SHIFT_LEFT_EQUAL**

Aseba::*Compiler*::*Token***TOKEN_OP_SHIFT_RIGHT_EQUAL**

Aseba::*Compiler*::*Token***TOKEN_OP_BIT_OR_EQUAL**

Aseba::*Compiler*::*Token***TOKEN_OP_BIT_XOR_EQUAL**

Aseba::*Compiler*::*Token***TOKEN_OP_BIT_AND_EQUAL**

Aseba::*Compiler*::*Token***TOKEN_OP_PLUS_PLUS**

Aseba::*Compiler*::*Token***TOKEN_OP_MINUS_MINUS**

Public Functions

Aseba::*Compiler*::*Token***Token**()

Aseba::Compiler::Token::Token(Type type, SourcePos pos = SourcePos (), const std::wstr
Construct a new token of given type and value.

const std::wstring *Aseba*::*Compiler*::*Token***typeName**() **const**
Return the name of the type of this token.

std::wstring *Aseba*::*Compiler*::*Token***toWString**() **const**
Return a string representation of the token.

Aseba::*Compiler*::*Token***operator Type**() **const**

Public Members

Aseba::*Compiler*::*Token*::*Type Aseba*::*Compiler*::*Token***TOKEN_END_OF_STREAM**
type of this token

std::wstring *Aseba*::*Compiler*::*Token***sValue**
string version of the value

int *Aseba*::*Compiler*::*Token***iValue** = {0}
int version of the value, 0 if not applicable

SourcePos Aseba::*Compiler*::*Token***pos**
position of token in source code

struct *Aseba***TranslatableError** : **public** *Aseba*::*Error*

Public Functions

Aseba::*TranslatableError* **TranslatableError**()

Aseba::*TranslatableError* **TranslatableError**(**const** *SourcePos* &*pos*, *ErrorCode* *error*)

TranslatableError &*Aseba*::*TranslatableError* **arg**(int *value*, int *fieldWidth* = 0, int *base* = 10, wchar_t *fillChar* = ' ')

TranslatableError &*Aseba*::*TranslatableError* **arg**(long int *value*, int *fieldWidth* = 0, int *base* = 10, wchar_t *fillChar* = ' ')

TranslatableError &*Aseba*::*TranslatableError* **arg**(unsigned *value*, int *fieldWidth* = 0, int *base* = 10, wchar_t *fillChar* = ' ')

TranslatableError &*Aseba*::*TranslatableError* **arg**(float *value*, int *fieldWidth* = 0, int *precision* = 6, wchar_t *fillChar* = ' ')

TranslatableError &*Aseba*::*TranslatableError* **arg**(**const** std::wstring &*value*)

Error *Aseba*::*TranslatableError* **toError**()

Public Members

WFormatableString *Aseba*::*TranslatableError* **message**

Public Static Functions

void *Aseba*::*TranslatableError* **setTranslateCB**(*ErrorMessages*::*ErrorCallback* *newCB*)

Public Static Attributes

ErrorMessages::*ErrorCallback* *Aseba*::*TranslatableError* **translateCB** = nullptr

struct *Aseba* **TupleVectorNode** : **public** *Aseba*::*AbstractTreeNode*

#include <tree.h> *Node* for assembling values into an array children[x] is the x-th *Node* to be assembled.

Public Functions

Aseba::*TupleVectorNode* **TupleVectorNode**(**const** *SourcePos* &*sourcePos*)

Constructor.

Aseba::*TupleVectorNode* **TupleVectorNode**(**const** *SourcePos* &*sourcePos*, int *value*)

TupleVectorNode **Aseba*::*TupleVectorNode* **shallowCopy**() **const**

Return a shallow copy of the object (children point to the same objects)

Node **Aseba*::*TupleVectorNode* **expandAbstractNodes**(std::wostream **dump*)

Dummy function, the node will be expanded during the vectorial pass.

Node **Aseba*::*TupleVectorNode* **expandVectorialNodes**(std::wostream **dump*, *Compiler* **compiler* = nullptr, unsigned int *index* = 0)

Expand to vector[index].

std::wstring *Aseba*::*TupleVectorNode*`toWString` () `const`
Return a string representation of this node.

std::wstring *Aseba*::*TupleVectorNode*`toNodeName` () `const`
Return a string representation of the name of this node.

void *Aseba*::*TupleVectorNode*`dump` (std::wostream &*dest*, unsigned &*indent*) `const`
Dump this node and the rest of the tree.

unsigned *Aseba*::*TupleVectorNode*`getVectorSize` () `const`
return the children's size, check for equal size, or E_NOVAL if no child

bool *Aseba*::*TupleVectorNode*`isImmediateVector` () `const`

int *Aseba*::*TupleVectorNode*`getImmediateValue` (unsigned *index*) `const`
return the n-th vector's element

`virtual` void *Aseba*::*TupleVectorNode*`addImmediateValue` (int *value*)

`struct` *Aseba*`UnaryArithmeticAssignmentNode` : `public` *Aseba*::*AbstractTreeNode*
#include <tree.h> *Node* for operations like "vector(op)", may be ++ or Will expand to "vector (op)= [1,. . . ,1]" children[0] is a *MemoryVectorNode*.

Public Functions

Aseba::*UnaryArithmeticAssignmentNode*`UnaryArithmeticAssignmentNode` (`const` *SourcePos* &*sourcePos*, *Compiler*::*Token*::*Type* *token*, *Node* **memory*)
Constructor.

UnaryArithmeticAssignmentNode **Aseba*::*UnaryArithmeticAssignmentNode*`shallowCopy` () `const`
Return a shallow copy of the object (children point to the same objects)

Node **Aseba*::*UnaryArithmeticAssignmentNode*`expandAbstractNodes` (std::wostream **dump*)
Expand "vector (opop)" to "vector (op)= 1", and then call for the expansion of this last expression.

Node **Aseba*::*UnaryArithmeticAssignmentNode*`expandVectorialNodes` (std::wostream **dump*, *Compiler* **compiler* = nullptr, unsigned int *index* = 0)
Third pass to expand vectorial operations into mutliple scalar ones.

Generic implementation for non-vectorial nodes.

std::wstring *Aseba*::*UnaryArithmeticAssignmentNode*`toWString` () `const`
Return a string representation of this node.

std::wstring *Aseba*::*UnaryArithmeticAssignmentNode*`toNodeName` () `const`
Return a string representation of the name of this node.

Public Members

AsebaBinaryOperator *Aseba*::*UnaryArithmeticAssignmentNode*`arithmeticOp`
operator

struct *Aseba***UnaryArithmeticNode** : **public** *Aseba*::*Node*

#include <tree.h> *Node* for unary arithmetic children[0] is the expression to negate.

Public Functions

Aseba::*UnaryArithmeticNode***UnaryArithmeticNode**(**const** *SourcePos* &*sourcePos*)

Constructor.

Aseba::*UnaryArithmeticNode***UnaryArithmeticNode**(**const** *SourcePos* &*sourcePos*, AsebaUnaryOperator *op*, *Node* *child*)

Constructor.

UnaryArithmeticNode **Aseba*::*UnaryArithmeticNode***shallowCopy**() **const**

Return a shallow copy of the object (children point to the same objects)

Node::*ReturnType Aseba*::*UnaryArithmeticNode***typeCheck**(*Compiler* *compiler)

Typecheck this node, throw an exception if there is any type violation.

Node **Aseba*::*UnaryArithmeticNode***optimize**(std::wostream *dump)

Optimize this node, return the optimized node.

void *Aseba*::*UnaryArithmeticNode***emit**(*PreLinkBytecode* &*bytecodes*) **const**

Generate bytecode.

std::wstring *Aseba*::*UnaryArithmeticNode***toWString**() **const**

Return a string representation of this node.

std::wstring *Aseba*::*UnaryArithmeticNode***toNodeName**() **const**

Return a string representation of the name of this node.

Public Members

AsebaUnaryOperator *Aseba*::*UnaryArithmeticNode***op**

operator

struct *Aseba***WhileNode** : **public** *Aseba*::*Node*

#include <tree.h> *Node* for L"while".

children[0] is expression children[1] is block

Public Functions

Aseba::*WhileNode***WhileNode**(**const** *SourcePos* &*sourcePos*)

Constructor.

WhileNode **Aseba*::*WhileNode***shallowCopy**() **const**

Return a shallow copy of the object (children point to the same objects)

void *Aseba*::*WhileNode***checkVectorSize**() **const**

Check the consistency in vectors' size.

Node::*ReturnType Aseba*::*WhileNode***typeCheck**(*Compiler* *compiler)

Typecheck this node, throw an exception if there is any type violation.

Node **Aseba*::*WhileNode***optimize**(std::wostream **dump*)
Optimize this node, return the optimized node.

void *Aseba*::*WhileNode***emit**(*PreLinkBytecode* &*bytecodes*) **const**
Generate bytecode.

std::wstring *Aseba*::*WhileNode***toWString**() **const**
Return a string representation of this node.

std::wstring *Aseba*::*WhileNode***toNodeName**() **const**
Return a string representation of the name of this node.

namespace Aseba

Typedefs

using Aseba::VariablesNamesVector = typedef std::vector<std::wstring>
Vector of names of variables.

using Aseba::EventDescription = typedef NamedValue
An event description is a name - value pair.

using Aseba::ConstantDefinition = typedef NamedValue
An constant definition is a name - value pair.

using Aseba::EventsDescriptionsVector = typedef NamedValuesVector
Vector of events descriptions.

using Aseba::ConstantsDefinitions = typedef NamedValuesVector
Vector of constants definitions.

Enums

enum *Aseba***ErrorCode**
Values:

*Aseba***ERROR_BROKEN_TARGET** = 0

*Aseba***ASEBA_ERROR_STACK_OVERFLOW**

*Aseba***ERROR_SCRIPT_TOO_BIG**

*Aseba***ERROR_VARIABLE_NOT_DEFINED**

*Aseba***ERROR_VARIABLE_NOT_DEFINED_GUESS**

*Aseba***ERROR_FUNCTION_NOT_DEFINED**

*Aseba***ERROR_FUNCTION_NOT_DEFINED_GUESS**

*Aseba***ERROR_CONSTANT_NOT_DEFINED**

*Aseba***ERROR_CONSTANT_NOT_DEFINED_GUESS**

*Aseba***ERROR_EVENT_NOT_DEFINED**

*Aseba***ERROR_EVENT_NOT_DEFINED_GUESS**

*Aseba***ERROR_EMIT_LOCAL_EVENT**

*Aseba***ERROR_SUBROUTINE_NOT_DEFINED**

*Aseba*ERROR_SUBROUTINE_NOT_DEFINED_GUESS

*Aseba*ERROR_LINE

*Aseba*ERROR_COL

*Aseba*ERROR_UNBALANCED_COMMENT_BLOCK

*Aseba*ERROR_SYNTAX

*Aseba*ERROR_INVALID_IDENTIFIER

*Aseba*ERROR_INVALID_HEXA_NUMBER

*Aseba*ERROR_INVALID_BINARY_NUMBER

*Aseba*ERROR_NUMBER_INVALID_BASE

*Aseba*ERROR_IN_NUMBER

*Aseba*ERROR_INTERNAL

*Aseba*ERROR_EXPECTING

*Aseba*ERROR_UINT12_OUT_OF_RANGE

*Aseba*ERROR_UINT16_OUT_OF_RANGE

*Aseba*ERROR_PINT16_OUT_OF_RANGE

*Aseba*ERROR_INT16_OUT_OF_RANGE

*Aseba*ERROR_PCONSTANT_OUT_OF_RANGE

*Aseba*ERROR_CONSTANT_OUT_OF_RANGE

*Aseba*ERROR_EXPECTING_ONE_OF

*Aseba*ERROR_NOT_ENOUGH_TEMP_SPACE

*Aseba*ERROR_MISPLACED_VARDEF

*Aseba*ERROR_EXPECTING_IDENTIFIER

*Aseba*ERROR_CONST_ALREADY_DEFINED

*Aseba*ERROR_VAR_ALREADY_DEFINED

*Aseba*ERROR_VAR_CONST_COLLISION

*Aseba*ERROR_UNDEFINED_SIZE

*Aseba*ERROR_NOT_ENOUGH_SPACE

*Aseba*ERROR_EXPECTING_ASSIGNMENT

*Aseba*ERROR_FOR_NULL_STEPS

*Aseba*ERROR_FOR_START_HIGHER_THAN_END

*Aseba*ERROR_FOR_START_LOWER_THAN_END

*Aseba*ERROR_FOR_INVALID_INC_END_INDEX

*Aseba*ERROR_FOR_INVALID_DEC_END_INDEX

*Aseba*ERROR_EVENT_ALREADY_IMPL

*Aseba*ERROR_EVENT_ARG_TOO_BIG

*Aseba*ERROR_EVENT_WRONG_ARG_SIZE

*Aseba*ERROR_SUBROUTINE_ALREADY_DEF

*Aseba*ERROR_INDEX_EXPECTING_CONSTANT

*Aseba*ERROR_INDEX_WRONG_END

*Aseba*ERROR_SIZE_IS_NEGATIVE

*Aseba*ERROR_SIZE_IS_NULL

*Aseba*ERROR_NOT_CONST_EXPR

*Aseba*ERROR_FUNCTION_HAS_NO_ARG

*Aseba*ERROR_FUNCTION_NO_ENOUGH_ARG

*Aseba*ERROR_FUNCTION_WRONG_ARG_SIZE

*Aseba*ERROR_FUNCTION_WRONG_ARG_SIZE_TEMPLATE

*Aseba*ERROR_FUNCTION_TOO_MANY_ARG

*Aseba*ERROR_UNARY_ARITH_BUILD_UNEXPECTED

*Aseba*ERROR_INCORRECT_LEFT_VALUE

*Aseba*ERROR_ARRAY_OUT_OF_BOUND

*Aseba*ERROR_ARRAY_SIZE_MISMATCH

*Aseba*ERROR_IF_VECTOR_CONDITION

*Aseba*ERROR_WHILE_VECTOR_CONDITION

*Aseba*ERROR_ARRAY_ILLEGAL_ACCESS

*Aseba*ERROR_INFINITE_LOOP

*Aseba*ERROR_DIVISION_BY_ZERO

*Aseba*ERROR_ABS_NOT_POSSIBLE

*Aseba*ERROR_ARRAY_OUT_OF_BOUND_READ

*Aseba*ERROR_ARRAY_OUT_OF_BOUND_WRITE

*Aseba*ERROR_EXPECTING_TYPE

*Aseba*ERROR_EXPECTING_CONDITION

*Aseba*ERROR_TOKEN_END_OF_STREAM

*Aseba*ERROR_TOKEN_STR_when

*Aseba*ERROR_TOKEN_STR_emit

*Aseba*ERROR_TOKEN_STR_hidden_emit

*Aseba*ERROR_TOKEN_STR_for

*Aseba*ERROR_TOKEN_STR_in

*Aseba*ERROR_TOKEN_STR_step

*Aseba*ERROR_TOKEN_STR_while

*Aseba*ERROR_TOKEN_STR_do

*Aseba*ERROR_TOKEN_STR_if

*Aseba*ERROR_TOKEN_STR_then

*Aseba*ERROR_TOKEN_STR_else

*Aseba*ERROR_TOKEN_STR_elseif

*Aseba*ERROR_TOKEN_STR_end

*Aseba*ERROR_TOKEN_STR_var

*Aseba*ERROR_TOKEN_STR_const

*Aseba*ERROR_TOKEN_STR_call

*Aseba*ERROR_TOKEN_STR_sub

*Aseba*ERROR_TOKEN_STR_callsub

*Aseba*ERROR_TOKEN_STR_onevent

*Aseba*ERROR_TOKEN_STR_abs

*Aseba*ERROR_TOKEN_STR_return

*Aseba*ERROR_TOKEN_STRING_LITERAL

*Aseba*ERROR_TOKEN_INT_LITERAL

*Aseba*ERROR_TOKEN_PAR_OPEN

*Aseba*ERROR_TOKEN_PAR_CLOSE

*Aseba*ERROR_TOKEN_BRACKET_OPEN

*Aseba*ERROR_TOKEN_BRACKET_CLOSE

*Aseba*ERROR_TOKEN_COLON

*Aseba*ERROR_TOKEN_COMMA

*Aseba*ERROR_TOKEN_ASSIGN

*Aseba*ERROR_TOKEN_OP_OR

*Aseba*ERROR_TOKEN_OP_AND

*Aseba*ERROR_TOKEN_OP_NOT

*Aseba*ERROR_TOKEN_OP_BIT_OR

*Aseba*ERROR_TOKEN_OP_BIT_XOR

*Aseba*ERROR_TOKEN_OP_BIT_AND

*Aseba*ERROR_TOKEN_OP_BIT_NOT

*Aseba*ERROR_TOKEN_OP_BIT_OR_EQUAL

*Aseba*ERROR_TOKEN_OP_BIT_XOR_EQUAL

*Aseba*ERROR_TOKEN_OP_BIT_AND_EQUAL

*Aseba*ERROR_TOKEN_OP_EQUAL

*Aseba*ERROR_TOKEN_OP_NOT_EQUAL

*Aseba*ERROR_TOKEN_OP_BIGGER

*Aseba*ERROR_TOKEN_OP_BIGGER_EQUAL

*Aseba*ERROR_TOKEN_OP_SMALLER

*Aseba*ERROR_TOKEN_OP_SMALLER_EQUAL

*Aseba***ERROR_TOKEN_OP_SHIFT_LEFT**

*Aseba***ERROR_TOKEN_OP_SHIFT_RIGHT**

*Aseba***ERROR_TOKEN_OP_SHIFT_LEFT_EQUAL**

*Aseba***ERROR_TOKEN_OP_SHIFT_RIGHT_EQUAL**

*Aseba***ERROR_TOKEN_OP_ADD**

*Aseba***ERROR_TOKEN_OP_NEG**

*Aseba***ERROR_TOKEN_OP_ADD_EQUAL**

*Aseba***ERROR_TOKEN_OP_NEG_EQUAL**

*Aseba***ERROR_TOKEN_OP_PLUS_PLUS**

*Aseba***ERROR_TOKEN_OP_MINUS_MINUS**

*Aseba***ERROR_TOKEN_OP_MULT**

*Aseba***ERROR_TOKEN_OP_DIV**

*Aseba***ERROR_TOKEN_OP_MOD**

*Aseba***ERROR_TOKEN_OP_MULT_EQUAL**

*Aseba***ERROR_TOKEN_OP_DIV_EQUAL**

*Aseba***ERROR_TOKEN_OP_MOD_EQUAL**

*Aseba***ERROR_TOKEN_UNKNOWN**

*Aseba***ERROR_UNKNOWN_ERROR**

*Aseba***ERROR_END**

Functions

template <class T>
unsigned int *Aseba***editDistance**(**const** T &*s1*, **const** T &*s2*, **const** unsigned *maxDist*)

Compute the edit distance between two vector-style containers, inspired from http://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Levenshtein_distance#C.2B.2B.

template <typename MapType>
MapType::const_iterator *Aseba***findInTable**(**const** MapType &*map*, **const** std::wstring &*name*, **const** *SourcePos* &*pos*, **const** *ErrorCode notFoundError*, **const** *ErrorCode misspelledError*)

Helper function to find for something in one of the map, using edit-distance to check for candidates if not found.

template <typename T>
bool *Aseba***isPOT**(T *number*)

Return true if number is a power of two.

template <typename T>
unsigned *Aseba***shiftFromPOT**(T *number*)

If number is a power of two, number = (1 << shift) and this function returns shift, otherwise return value is undefined.

std::wstring *Aseba***binaryOperatorToString**(AsebaBinaryOperator *op*)

Return the string corresponding to the binary operator.

std::wstring *Aseba*::**unaryOperatorToString** (AsebaUnaryOperator *op*)
Return the string corresponding to the unary operator.

static void *Aseba*::**addImmediateToBytecodes** (int *value*, **const** *SourcePos* &*sourcePos*, *PreLinkBytecode* &*bytecodes*)

static bool *Aseba*::**matchNameInMemoryVector** (**const** *Node* **root*, std::wstring *name*)

Variables

const wchar_t **Aseba*::**error_map**[**ERROR_END**]

file **analysis.cpp**
#include "compiler.h"#include "common/consts.h"#include <cassert>#include <iostream>

file **compiler.cpp**
#include "compiler.h"#include "tree.h"#include "errors_code.h"#include "common/consts.h"#include "common/utils/utils.h"#include "common/utils/FormatableString.h"#include <cassert>#include <cstdlib>#include <sstream>#include <iostream>#include <fstream>#include <iomanip>#include <memory>#include <limits>#include <iterator>

file **compiler.h**
#include <utility>#include <vector>#include <deque>#include <string>#include <map>#include <set>#include <istream>#include "errors_code.h"#include "common/types.h"#include "common/msg/TargetDescription.h"#include "common/utils/FormatableString.h"

file **errors.cpp**
#include "errors_code.h"#include "compiler.h"#include <sstream>#include <string>

file **errors_code.h**

file **identifier-lookup.cpp**
#include "compiler.h"#include "tree.h"#include "common/consts.h"#include "common/utils/FormatableString.h"#include <cassert>#include <cstdlib>#include <sstream>#include <iostream>#include <fstream>#include <iomanip>#include <memory>#include <limits>#include <algorithm>#include <malloc.h> Functions for quick lookup of identifiers (variables, events, subroutines, native functions, constants) using maps.

file **lexer.cpp**
#include "compiler.h"#include "common/utils/FormatableString.h"#include "common/utils/utils.h"#include <cstdlib>#include <sstream>#include <ostream>#include <cctype>#include <cstdio>#include <cwctype>#include <locale>

file **mainpage.dox**

file **natives.c**
#include "common/consts.h"#include "common/types.h"#include "natives.h"#include <string.h>#include <assert.h>

Functions

int16_t **aseba_atan2** (int16_t *y*, int16_t *x*)

int16_t **aseba_sin** (int16_t *angle*)

int16_t **aseba_cos** (int16_t *angle*)

int16_t **aseba_sqrt** (int16_t *num*)

void **aseba_comb_sort** (int16_t **input*, uint16_t *size*)

void **AsebaNative_veccopy** (*AsebaVMState* **vm*)
Function to copy a vector.

void **AsebaNative_vecfill** (*AsebaVMState* **vm*)
Function to fill all the elements of a vector to a specific value.

void **AsebaNative_vecaddscalar** (*AsebaVMState* **vm*)
Function to add a scalar to each element of a vector.

void **AsebaNative_vecadd** (*AsebaVMState* **vm*)
Function to add two vectors.

void **AsebaNative_vecsub** (*AsebaVMState* **vm*)
Function to substract two vectors.

void **AsebaNative_vecmul** (*AsebaVMState* **vm*)
Function to multiply two vectors elements by elements.

void **AsebaNative_vecdiv** (*AsebaVMState* **vm*)
Function to divide two vectors elements by elements.

void **AsebaNative_vecmin** (*AsebaVMState* **vm*)
Function to take the element by element minimum.

void **AsebaNative_vecmax** (*AsebaVMState* **vm*)
Function to take the element by element maximum.

void **AsebaNative_vecclamp** (*AsebaVMState* **vm*)
Function to clamp a vector of values element by element.

void **AsebaNative_vecdot** (*AsebaVMState* **vm*)
Function to perform a dot product on a vector.

void **AsebaNative_vecstat** (*AsebaVMState* **vm*)
Function to perform statistics on a vector.

void **AsebaNative_vecargbounds** (*AsebaVMState* **vm*)
Function to get indices of the bounds of a vector.

void **AsebaNative_vecsort** (*AsebaVMState* **vm*)
Function to sort a vector.

void **AsebaNative_mathmuldiv** (*AsebaVMState* **vm*)
Function to perform dest = (a*b)/c in 32 bits.

void **AsebaNative_mathatan2** (*AsebaVMState* **vm*)
Function to perform atan2.

void **AsebaNative_mathsin** (*AsebaVMState* **vm*)
Function to perform sin.

void **AsebaNative_mathcos** (*AsebaVMState* **vm*)
Function to perform cos.

void **AsebaNative_mathrot2** (*AsebaVMState* **vm*)
Function to perform the rotation of a vector.

void **AsebaNative_mathsqrt** (*AsebaVMState* **vm*)
Function to perform sqrt.

void **AsebaNative_vecnonzerosequence** (*AsebaVMState* **vm*)
Function to get the middle index of the largest sequence of non-zero elements.

void **AsebaSetRandomSeed**(uint16_t *seed*)
Functon to set the seed of random generator.

uint16_t **AsebaGetRandom**()
Functon to get a random number.

void **AsebaNative_rand**(*AsebaVMState* *vm)
Function to get a 16-bit signed random number.

static void **deque_shift**(*AsebaVMState* *vm, uint16_t *dq*, uint16_t *dq_capacity*, uint16_t *target*, int16_t *last*, int16_t *delta*)

static void **deque_throw_exception**(*AsebaVMState* *vm)

void **AsebaNative_deqsize**(*AsebaVMState* *vm)
Function that reports size of deque dest in size.

static void **_AsebaNative_deqget**(*AsebaVMState* *vm, uint16_t *dest*, uint16_t *deque*, uint16_t *index_val*, uint16_t *dest_length*, uint16_t *deque_length*)

void **AsebaNative_deqget**(*AsebaVMState* *vm)
Function that copies index-th len-dest elements of deque src to dest.

void **AsebaNative_deqset**(*AsebaVMState* *vm)
Function that copies len-src elements of src to index-th position of deque dest.

static void **_AsebaNative_deqinsert**(*AsebaVMState* *vm, uint16_t *deque*, uint16_t *src*, uint16_t *index_val*, uint16_t *deque_length*, uint16_t *src_length*)

void **AsebaNative_deqinsert**(*AsebaVMState* *vm)
Function that copies len-src elements of src before index-th position of deque dest.

static void **_AsebaNative_deqerase**(*AsebaVMState* *vm, uint16_t *deque*, uint16_t *index_val*, uint16_t *len_val*, uint16_t *deque_length*)

void **AsebaNative_deqerase**(*AsebaVMState* *vm)
Function that erases len elements from deque dest starting from index-th position.

void **AsebaNative_deqpushfront**(*AsebaVMState* *vm)
Function that copies len-src elements of src before the front of deque dest.

void **AsebaNative_deqpushback**(*AsebaVMState* *vm)
Function that copies len-src elements of src after the end of deque dest.

void **AsebaNative_deqpopfront**(*AsebaVMState* *vm)
Function that erases len elements from deque dest starting at the front.

void **AsebaNative_deqpopback**(*AsebaVMState* *vm)
Function that erases len elements from deque dest starting from the end.

Variables

652, 735, 816, 896, 977, 1058, 1139, 1218, 1300, 1459, 1620, 1777, 1935, 2093, 2250, 2403, 2556, 2868, 3164, 3458, 3748, 4029, 4307, 4578, 4839, 5359, 5836, 6290, 6720, 7126, 7507, 7861, 8203, 8825, 9357, 9839, 10260, 10640, 10976, 11281, 11557, 12037, 12425, 12755, 13036, 13277, 13486, 13671, 13837, 14112, 14331, 14514, 14666, 14796, 14907, 15003, 15091, 15235, 15348, 15441, 15519, 15585, 15642, 15691, 15736, 15808, 15865, 15912, 15951, 15984, 16013, 16037, 16060, 16096, 16125, 16148, 16168, 16184, 16199, 16211, 16222, 16240, 16255, 16266, 16276, 16284, 16292, 16298, 16303, 16312, 16320, 16325, 16331, 16334, 16338, 16341, 16344, 16348, 16352, 16355, 16357, 16360, 16361, 16363, 16364, 16366, 16369, 16369, 16371, 16372, 16373, 16373, 16375, 16375, 16377, 16376, 16378, 16378, 16378, 16379, 16379, 16380, 16380, 16380, 16382, 16381, 16381, 16381, 16382, 16382, 16382, 16382, 16382,

16382, 16384, 16383, 16383, 16383, 16383, 16383, 16383, 16383, 16383, 16383, 16383, 16383, 16384, 16384, 16384, 16384, 16384, 16384, 16384, 16384, 16384, 16384, 16384, 16384, 16384, 16384}]
0, 403, 804, 1207, 1608, 2010, 2411, 2812, 3212, 3612, 4011, 4411, 4808, 5206, 5603, 5998, 6393, 6787, 7180, 7572, 7962, 8352, 8740, 9127, 9513, 9896, 10279, 10660, 11040, 11417, 11794, 12167, 12540, 12911, 13279, 13646, 14010, 14373, 14733, 15091, 15447, 15801, 16151, 16500, 16846, 17190, 17531, 17869, 18205, 18538, 18868, 19196, 19520, 19842, 20160, 20476, 20788, 21097, 21403, 21706, 22006, 22302, 22595, 22884, 23171, 23453, 23732, 24008, 24279, 24548, 24812, 25073, 25330, 25583, 25833, 26078, 26320, 26557, 26791, 27020, 27246, 27467, 27684, 27897, 28106, 28311, 28511, 28707, 28899, 29086, 29269, 29448, 29622, 29792, 29957, 30117, 30274, 30425, 30572, 30715, 30852, 30985, 31114, 31238, 31357, 31471, 31581, 31686, 31786, 31881, 31972, 32057, 32138, 32215, 32285, 32352, 32413, 32470, 32521, 32569, 32610, 32647, 32679, 32706, 32728, 32746, 32758, 32766, 32767, }]
“copies src to dest element by element”, {{-1, “dest”}, {-1, “src”}, {0, 0}}}]Description of AsebaNative_veccopy.
“fills dest with constant value”, {{-1, “dest”}, {1, “value”}, {0, 0}}}]Description of AsebaNative_vecfill.
“math.addscalar”, “add scalar to each element of src and store result to dest”, {{-1, “dest”}, {-1, “src”}, {1, “scalar”}, {0, 0}}}]Description of AsebaNative_vecaddscalar.
“math.add”, “adds src1 and src2, write to dest, element by element”, {{-1, “dest”}, {-1, “src1”}, {-1, “src2”}, {0, 0}}}]Description of AsebaNative_vecadd.
“math.sub”, “substracts src2 from src1 to dest, element by element”, {{-1, “dest”}, {-1, “src1”}, {-1, “src2”}, {0, 0}}}]Description of AsebaNative_vecsub.
“math.mul”, “multiplies src1 and src2 to dest, element by element”, {{-1, “dest”}, {-1, “src1”}, {-1, “src2”}, {0, 0}}}]Description of AsebaNative_vecadd.
“math.div”, “divides src1 by src2 to dest, element by element”, {{-1, “dest”}, {-1, “src1”}, {-1, “src2”}, {0, 0}}}]Description of AsebaNative_vecsub.
“math.min”, “writes the minimum of src1 and src2 to dest, element by element”, {{-1, “dest”}, {-1, “src1”}, {-1, “src2”}, {0, 0}}}]Description of AsebaNative_vecmin.
“math.max”, “writes the maximum of src1 and src2 to dest, element by element”, {{-1, “dest”}, {-1, “src1”}, {-1, “src2”}, {0, 0}}}]Description of AsebaNative_vecsmax.
“math.clamp”, “clamp src to low/high bounds into dest, element by element”, {{-1, “dest”}, {-1, “src”}, {-1, “low”}, {-1, “high”}, {0, 0}}}]Description of AsebaNative_vecclamp.
“math.dot”, “writes the dot product of src1 and src2 to dest, after a shift”, {{1, “dest”}, {-1, “src1”}, {-1, “src2”}, {1, “shift”}, {0, 0}}}]Description of AsebaNative_vecdot.
“math.stat”, “performs statistics on src”, {{-1, “src”}, {1, “min”}, {1, “max”}, {1, “mean”}, {0, 0}}}]Description of AsebaNative_vecstat.
“math.argbounds”, “get the indices (argmin, argmax) of the limit values of src”, {{-1, “src”}, {1, “argmin”}, {1, “argmax”}, {0, 0}}}]Description of AsebaNative_vecargbounds.
“sort array in place”, {{-1, “array”}, {0, 0}}}]Description of AsebaNative_vecsort.
“math.muldiv”, “performs dest = (a*b)/c in 32 bits element by element”, {{-1, “dest”}, {-1, “a”}, {-1, “b”}, {-1, “c”}, {0, 0}}}]Description of AsebaNative_mathmuldiv.
“math.atan2”, “performs dest = atan2(y,x) element by element”, {{-1, “dest”}, {-1, “y”}, {-1, “x”}, {0, 0}}}]Description of AsebaNative_mathatan2.
“performs dest = sin(x) element by element”, {{-1, “dest”}, {-1, “x”}, {0, 0}}}]Description of AsebaNative_mathsin.
“performs dest = cos(x) element by element”, {{-1, “dest”}, {-1, “x”}, {0, 0}}}]Description of AsebaNative_mathcos.
“rotates v of angle a to dest”, {{2, “dest”}, {2, “v”}, {1, “a”}, {0, 0}}}]Description of AsebaNative_mathrot2.
“performs dest = sqrt(x) element by element”, {{-1, “dest”}, {-1, “x”}, {0, 0}}}]Description of AsebaNative_mathsqrt.
“math.nzseq”, “write to dest the middle index of the largest sequence of non-zero elements from src, -1 if “ “not found or if smaller than minLength”, {{1, “dest”}, {-1, “src”}, {1, “minLength”}, {0, 0}}}]Description of AsebaNative_vecnonzerosequence.

uint16_t **rnd_state**

"fill array with random values", {{-1, "dest"}, {0, 0}}}]Description of AsebaNative_rand.
"report size of dest", {{-1, "deque"}, {1, "size"}, {0, 0}}}]Description of AsebaNative_deqsize.
"deque.get", "fill dest from deque at index", {{-1, "deque"}, {-2, "dest"}, {1, "index"}, {0, 0}}}]Description of AsebaNative_deqget.
"deque.set", "copies src to deque at index", {{-1, "deque"}, {-2, "src"}, {1, "index"}, {0, 0}}}]Description of AsebaNative_deqset.
"deque.insert", "shift deque at index by len(src) and insert src", {{-1, "deque"}, {-2, "src"}, {1, "index"}, {0, 0}}}]Description of AsebaNative_deqinsert.
"deque.erase", "shift deque at index by len to erase", {{-1, "deque"}, {1, "index"}, {1, "len"}, {0, 0}}}]Description of AsebaNative_deqerase.
"insert src before the front of deque", {{-1, "deque"}, {-2, "src"}, {0, 0}}}]Description of AsebaNative_deqpushfront.
"insert src after the back of deque", {{-1, "deque"}, {-2, "src"}, {0, 0}}}]Description of AsebaNative_deqpushback.
"deque.pop_front", "fill dest from front of deque then erase elements", {{-1, "deque"}, {-2, "dest"}, {0, 0}}}]Description of AsebaNative_deqpopfront.
"deque.pop_back", "fill dest from back of deque then erase elements", {{-1, "deque"}, {-2, "dest"}, {0, 0}}}]Description of AsebaNative_deqpopback.

file **natives.h**

#include "common/types.h"#include "vm.h" Definition of standard natives functions for Aseba Virtual Machine.

Defines

ASEBA_NATIVES_STD_COUNT

Embedded targets must know the size of ASEBA_NATIVES_STD_FUNCTIONS without having to compute them by hand, please update this when adding a new function.

ASEBA_NATIVES_STD_FUNCTIONS

snippet to include standard native functions

ASEBA_NATIVES_STD_DESCRIPTIONS

snippet to include descriptions of standard native functions

Typedefs

typedef void (***AsebaNativeFunctionPointer**) (*AsebaVMState* *vm)

Signature of a native function.

Functions

static int16_t **AsebaNativePopArg** (*AsebaVMState* **vm*)

Return an argument on the stack, including the value of template parameters.

void **AsebaNative_veccopy** (*AsebaVMState* **vm*)

Function to copy a vector.

void **AsebaNative_vecfill** (*AsebaVMState* **vm*)

Function to fill all the elements of a vector to a specific value.

void **AsebaNative_vecaddscalar** (*AsebaVMState* **vm*)

Function to add a scalar to each element of a vector.

void **AsebaNative_vecadd**(*AsebaVMState* *vm)
Function to add two vectors.

void **AsebaNative_vecsub**(*AsebaVMState* *vm)
Function to substract two vectors.

void **AsebaNative_vecmul**(*AsebaVMState* *vm)
Function to multiply two vectors elements by elements.

void **AsebaNative_vecdiv**(*AsebaVMState* *vm)
Function to divide two vectors elements by elements.

void **AsebaNative_vecmin**(*AsebaVMState* *vm)
Function to take the element by element minimum.

void **AsebaNative_vecmax**(*AsebaVMState* *vm)
Function to take the element by element maximum.

void **AsebaNative_vecclamp**(*AsebaVMState* *vm)
Function to clamp a vector of values element by element.

void **AsebaNative_vecdot**(*AsebaVMState* *vm)
Function to perform a dot product on a vector.

void **AsebaNative_vecstat**(*AsebaVMState* *vm)
Function to perform statistics on a vector.

void **AsebaNative_vecargbounds**(*AsebaVMState* *vm)
Function to get indices of the bounds of a vector.

void **AsebaNative_vecsort**(*AsebaVMState* *vm)
Function to sort a vector.

void **AsebaNative_mathmuldiv**(*AsebaVMState* *vm)
Function to perform dest = (a*b)/c in 32 bits.

void **AsebaNative_mathatan2**(*AsebaVMState* *vm)
Function to perform atan2.

void **AsebaNative_mathsin**(*AsebaVMState* *vm)
Function to perform sin.

void **AsebaNative_mathcos**(*AsebaVMState* *vm)
Function to perform cos.

void **AsebaNative_mathrot2**(*AsebaVMState* *vm)
Function to perform the rotation of a vector.

void **AsebaNative_mathsqrt**(*AsebaVMState* *vm)
Function to perform sqrt.

void **AsebaNative_vecnonzerosequence**(*AsebaVMState* *vm)
Function to get the middle index of the largest sequence of non-zero elements.

void **AsebaSetRandomSeed**(uint16_t *seed*)
Functon to set the seed of random generator.

uint16_t **AsebaGetRandom**()
Functon to get a random number.

void **AsebaNative_rand**(*AsebaVMState* *vm)
Function to get a 16-bit signed random number.

void **AsebaNative_deqsize** (*AsebaVMState* *vm)
Function that reports size of deque dest in size.

void **AsebaNative_deqget** (*AsebaVMState* *vm)
Function that copies index-th len-dest elements of deque src to dest.

void **AsebaNative_deqset** (*AsebaVMState* *vm)
Function that copies len-src elements of src to index-th position of deque dest.

void **AsebaNative_deqinsert** (*AsebaVMState* *vm)
Function that copies len-src elements of src before index-th position of deque dest.

void **AsebaNative_deqerase** (*AsebaVMState* *vm)
Function that erases len elements from deque dest starting from index-th position.

void **AsebaNative_deqpushfront** (*AsebaVMState* *vm)
Function that copies len-src elements of src before the front of deque dest.

void **AsebaNative_deqpushback** (*AsebaVMState* *vm)
Function that copies len-src elements of src after the end of deque dest.

void **AsebaNative_deqpopfront** (*AsebaVMState* *vm)
Function that erases len elements from deque dest starting at the front.

void **AsebaNative_deqpopback** (*AsebaVMState* *vm)
Function that erases len elements from deque dest starting from the end.

Variables

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_veccopy**
Description of AsebaNative_veccopy.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecfill**
Description of AsebaNative_vecfill.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecaddscalar**
Description of AsebaNative_vecaddscalar.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecadd**
Description of AsebaNative_vecadd.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecsub**
Description of AsebaNative_vecsub.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecmul**
Description of AsebaNative_vecadd.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecdiv**
Description of AsebaNative_vecsub.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecmin**
Description of AsebaNative_vecmin.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecmax**
Description of AsebaNative_vecsmax.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecclamp**
Description of AsebaNative_vecclamp.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecdot**
Description of AsebaNative_vecdot.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecstat**
Description of AsebaNative_vecstat.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecargbounds**
Description of AsebaNative_vecargbounds.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecsort**
Description of AsebaNative_vecsort.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_mathmuldiv**
Description of AsebaNative_mathmuldiv.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_mathatan2**
Description of AsebaNative_mathatan2.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_mathsin**
Description of AsebaNative_mathsin.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_mathcos**
Description of AsebaNative_mathcos.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_mathrot2**
Description of AsebaNative_mathrot2.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_mathsqrt**
Description of AsebaNative_mathsqrt.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecnonzerosequence**
Description of AsebaNative_vecnonzerosequence.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_rand**
Description of AsebaNative_rand.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_deqsize**
Description of AsebaNative_deqsize.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_deqget**
Description of AsebaNative_deqget.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_deqset**
Description of AsebaNative_deqset.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_deqinsert**
Description of AsebaNative_deqinsert.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_deqerase**
Description of AsebaNative_deqerase.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_deqpushfront**
Description of AsebaNative_deqpushfront.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_deqpushback**
Description of AsebaNative_deqpushback.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_deqpopfront**
Description of AsebaNative_deqpopfront.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_deqpopback**
Description of AsebaNative_deqpopback.

file **parser.cpp**
#include "compiler.h"#include "tree.h"#include "common/utils/FormatableString.h"#include "common/utils/utils.h"#include <memory>#include <valarray>#include <iostream>#include <cassert>#include <typeinfo>#include <algorithm>

Defines

IS_ONE_OF (array)

EXPECT_ONE_OF (array)

STRICT

file **power-of-two.h**

file **tree-build.cpp**
#include "tree.h"#include <cstdlib>#include <iostream>#include <utility>

file **tree-dump.cpp**
#include "tree.h"#include "common/utils/FormatableString.h"#include <ostream>

file **tree-emit.cpp**
#include "tree.h"#include "common/utils/utils.h"#include <cassert>#include <cstdlib>#include <algorithm>#include <iterator>

file **tree-expand.cpp**
#include "compiler.h"#include "tree.h"#include "common/utils/utils.h"#include "common/utils/FormatableString.h"#include <cassert>#include <memory>#include <iostream>

file **tree-optimize.cpp**
#include "tree.h"#include "power-of-two.h"#include "common/utils/FormatableString.h"#include "common/utils/utils.h"#include <cassert>#include <cstdlib>

file **tree-typecheck.cpp**
#include "tree.h"#include "common/utils/FormatableString.h"#include <cstdlib>

file **tree.h**
#include "compiler.h"#include "common/consts.h"#include "common/utils/FormatableString.h"#include <vector>#include <string>#include <ostream>#include <climits>#include <cassert>#include <iostream>

file **vm.c**
#include "common/consts.h"#include "common/types.h"#include "vm.h"#include <string.h> Implementation of standard natives functions for Aseba Virtual Machine.

Implementation of Aseba Virtual Machine.

Defines

GET_BIT (v, b)
Return true if bit b of v is 1.

BIT_SET (v, b)
Set bit b of v to 1.

BIT_CLR (v, b)
Set bit b of v to 0.

Functions

void **AsebaVMSendExecutionStateChanged** (*AsebaVMState* **vm*)
Send an execution state changed message.

void **AsebaVMInit** (*AsebaVMState* *vm)
Setup the execution status of the VM.

This is not sufficient to have a working VM. nodeId and bytecode, variables, and stack along with their sizes must be set outside this function. The content of the variable array is zeroed by this function.

uint16_t **AsebaVMGetEventAddress** (*AsebaVMState* *vm, uint16_t *event*)
Return the starting address of an event, or 0 if the event is not handled.

uint16_t **AsebaVMSetupEvent** (*AsebaVMState* *vm, uint16_t *event*)
Setup VM to execute an event.

If event is not handled, VM is not ready for run. Return the starting address of the event, or 0 if the event is not handled.

static int16_t **AsebaVMDoBinaryOperation** (*AsebaVMState* *vm, int16_t *valueOne*, int16_t *valueTwo*, uint16_t *op*)

static int16_t **AsebaVMDoUnaryOperation** (*AsebaVMState* *vm, int16_t *value*, uint16_t *op*)

void **AsebaVMStep** (*AsebaVMState* *vm)
Execute one bytecode of the current VM thread.

VM must be ready for run otherwise trashes may occur.

void **AsebaVMEmitNodeSpecificError** (*AsebaVMState* *vm, **const** char *message)
Can be called by glue code (including native functions), to stop vm and emit a node specific error.

uint16_t **AsebaVMCheckBreakpoint** (*AsebaVMState* *vm)
Execute on bytecode of the current VM thread and check for potential breakpoints.

Return 1 if breakpoint was seen, 0 otherwise. VM must be ready for run otherwise trashes may occur.

void **AsebaDebugBareRun** (*AsebaVMState* *vm, uint16_t *stepsLimit*)
Run without support of breakpoints.

Check ASEBA_VM_EVENT_RUNNING_MASK to exit on interrupts or stepsLimit if > 0.

void **AsebaDebugBreakpointRun** (*AsebaVMState* *vm, uint16_t *stepsLimit*)
Run with support of breakpoints.

Also check ASEBA_VM_EVENT_RUNNING_MASK to exit on interrupts.

uint16_t **AsebaVMRun** (*AsebaVMState* *vm, uint16_t *stepsLimit*)
Run the VM depending on the current execution mode.

Either run or step, depending of the current mode. If stepsLimit > 0, execute at maximim stepsLimit Return 1 if anything was executed, 0 otherwise.

uint8_t **AsebaVMSetBreakpoint** (*AsebaVMState* *vm, uint16_t *pc*)
Set a breakpoint at a specific location.

uint16_t **AsebaVMClearBreakpoint** (*AsebaVMState* *vm, uint16_t *pc*)
Clear the breakpoint at a specific location.

void **AsebaVMClearBreakpoints** (*AsebaVMState* *vm)
Clear all breakpoints.

static void **AsebaVMResetWhenFlags** (*AsebaVMState* *vm)
Reset all when flags in their default states in the bytecode.

void **AsebaVMDebugMessage** (*AsebaVMState* *vm, uint16_t *id*, uint16_t *data, uint16_t *dataLength*)
Execute a debug action from a debug message.

dataLength is given in number of uint16_t.

uint16_t **AsebaVMShouldDropPacket** (*AsebaVMState* *vm, uint16_t *source*, **const** uint8_t *data)
Return non-zero if VM will ignore the packet, 0 otherwise.

file **vm.h**
#include "common/types.h" Definition of Aseba Virtual Machine.

Defines

AsebaMaskSet (v, m)
Set the part masked by m of v to 1.

AsebaMaskClear (v, m)
Set the part masked by m of v to 0.

AsebaMaskIsSet (v, m)
Returns true if the part masked by m of v is 1.

AsebaMaskIsClear (v, m)
Returns true if the part masked by m of v is 0.

AsebaSendMessageWords (vm, type, data, size)

Enums

enum [anonymous]
Values:

ASEBA_MAX_BREAKPOINTS = 16
maximum number of simultaneous breakpoints the target supports

enum AsebaAssertReason
Possible causes of AsebaAssert.

Values:

ASEBA_ASSERT_UNKNOWN = 0

ASEBA_ASSERT_UNKNOWN_UNARY_OPERATOR

ASEBA_ASSERT_UNKNOWN_BINARY_OPERATOR

ASEBA_ASSERT_UNKNOWN_BYTECODE

ASEBA_ASSERT_STACK_OVERFLOW

ASEBA_ASSERT_STACK_UNDERFLOW

ASEBA_ASSERT_OUT_OF_VARIABLES_BOUNDS

ASEBA_ASSERT_OUT_OF_BYTECODE_BOUNDS

ASEBA_ASSERT_STEP_OUT_OF_RUN

ASEBA_ASSERT_BREAKPOINT_OUT_OF_BYTECODE_BOUNDS

ASEBA_ASSERT_EMIT_BUFFER_TOO_LONG

Functions

void **AsebaVMInit** (*AsebaVMState* *vm)
Setup the execution status of the VM.

This is not sufficient to have a working VM. nodeId and bytecode, variables, and stack along with their sizes must be set outside this function. The content of the variable array is zeroed by this function.

uint16_t **AsebaVMGetEventAddress** (*AsebaVMState* *vm, uint16_t *event*)
Return the starting address of an event, or 0 if the event is not handled.

uint16_t **AsebaVMSetupEvent** (*AsebaVMState* *vm, uint16_t *event*)
Setup VM to execute an event.

If event is not handled, VM is not ready for run. Return the starting address of the event, or 0 if the event is not handled.

uint16_t **AsebaVMRun** (*AsebaVMState* *vm, uint16_t *stepsLimit*)
Run the VM depending on the current execution mode.

Either run or step, depending of the current mode. If stepsLimit > 0, execute at maximim stepsLimit Return 1 if anything was executed, 0 otherwise.

void **AsebaVMDebugMessage** (*AsebaVMState* *vm, uint16_t *id*, uint16_t **data*, uint16_t *dataLength*)
Execute a debug action from a debug message.

dataLength is given in number of uint16_t.

void **AsebaVMEmitNodeSpecificError** (*AsebaVMState* *vm, **const** char **message*)
Can be called by glue code (including native functions), to stop vm and emit a node specific error.

uint16_t **AsebaVMShouldDropPacket** (*AsebaVMState* *vm, uint16_t *source*, **const** uint8_t **data*)
Return non-zero if VM will ignore the packet, 0 otherwise.

void **AsebaSendMessage** (*AsebaVMState* *vm, uint16_t *id*, **const** void **data*, uint16_t *size*)
Called by AsebaStep if there is a message (not an user event) to send.

size is given in number of bytes.

void **AsebaSendVariables** (*AsebaVMState* *vm, uint16_t *start*, uint16_t *length*)
Called by AsebaVMDebugMessage when some variables must be sent efficiently.

void **AsebaSendDescription** (*AsebaVMState* *vm)
Called by AsebaVMDebugMessage when VM must send its description on the network.

void **AsebaNativeFunction** (*AsebaVMState* *vm, uint16_t *id*)
Called by AsebaStep to perform a native function call.

void **AsebaWriteBytecode** (*AsebaVMState* *vm)
Called by AsebaVMDebugMessage when VM must write its bytecode to flash, write an empty function to leave feature unsupported.

void **AsebaResetIntoBootloader** (*AsebaVMState* *vm)
Called by AsebaVMDebugMessage when VM must restart the node and enter to the bootloader, write an empty function to leave feature unsupported.

void **AsebaPutVmToSleep** (*AsebaVMState* *vm)
Called by AsebaVMDebugMessage when VM must put to node in deep sleep.

Write an empty function to leave feature unsupported

void __attribute__((weak))
Called by AsebaVMDebugMessage when VM is going to be run.

Called by AsebaVMEmitNodeSpecificError to be notified when VM hit an execution error Is also called for wrong array access or division by 0 with message == NULL.

Called by AsebaVMDebugMessage when VM is being resetted.

void **AsebaAssert**(*AsebaVMState* *vm, *AsebaAssertReason* reason)
If ASEBA_ASSERT is defined, this function is called when an error arise.

Variables

void const char* message

group **compiler**

Typedefs

using Aseba::VariablesNamesVector = typedef std::vector<std::wstring>
Vector of names of variables.

using Aseba::EventDescription = typedef NamedValue
An event description is a name - value pair.

using Aseba::ConstantDefinition = typedef NamedValue
An constant definition is a name - value pair.

using Aseba::EventsDescriptionsVector = typedef NamedValuesVector
Vector of events descriptions.

using Aseba::ConstantsDefinitions = typedef NamedValuesVector
Vector of constants definitions.

Variables

const wchar_t *Aseba**error_map**[**ERROR_END**]

ErrorMessages::*ErrorCallback Aseba*::*TranslatableError***translateCB** = nullptr
ASEBA_OP_ADD, ASEBA_OP_SUB, ASEBA_OP_MULT, ASEBA_OP_DIV, ASEBA_OP_MOD, ASEBA_OP_SHIFT_LEFT, ASEBA_OP_SHIFT_RIGHT, ASEBA_OP_BIT_OR, ASEBA_OP_BIT_XOR, ASEBA_OP_BIT_AND }]

group **vm**
Glue logic must implement:

```
if (debug command queue is not empty)
    execute debug command
else if (executing a thread)
    run VM
else if (event queue is not empty)
    fetch event
    run VM
else
    sleep until something happens
```

Defines

ASEBA_NATIVES_STD_COUNT
Embedded targets must know the size of ASEBA_NATIVES_STD_FUNCTIONS without having to compute them by hand, please update this when adding a new function.

ASEBA_NATIVES_STD_FUNCTIONS
snippet to include standard native functions

ASEBA_NATIVES_STD_DESCRIPTIONS
snippet to include descriptions of standard native functions

GET_BIT(v, b)
Return true if bit b of v is 1.

BIT_SET(v, b)
Set bit b of v to 1.

BIT_CLR(v, b)
Set bit b of v to 0.

AsebaMaskSet(v, m)
Set the part masked by m of v to 1.

AsebaMaskClear(v, m)
Set the part masked by m of v to 0.

AsebaMaskIsSet(v, m)
Returns true if the part masked by m of v is 1.

AsebaMaskIsClear(v, m)
Returns true if the part masked by m of v is 0.

AsebaSendMessageWords(vm, type, data, size)

Typedefs

typedef void (***AsebaNativeFunctionPointer**) (*AsebaVMState* *vm)
Signature of a native function.

Enums

enum [anonymous]__anonymous0
Values:

vm**ASEBA_MAX_BREAKPOINTS** = 16
maximum number of simultaneous breakpoints the target supports

enum vm**AsebaAssertReason**
Possible causes of AsebaAssert.

Values:

vm**ASEBA_ASSERT_UNKNOWN** = 0

vm**ASEBA_ASSERT_UNKNOWN_UNARY_OPERATOR**

vm**ASEBA_ASSERT_UNKNOWN_BINARY_OPERATOR**

vm**ASEBA_ASSERT_UNKNOWN_BYTECODE**

vm**ASEBA_ASSERT_STACK_OVERFLOW**

vm**ASEBA_ASSERT_STACK_UNDERFLOW**

vm**ASEBA_ASSERT_OUT_OF_VARIABLES_BOUNDS**

vm**ASEBA_ASSERT_OUT_OF_BYTECODE_BOUNDS**

vm**ASEBA_ASSERT_STEP_OUT_OF_RUN**

vm**ASEBA_ASSERT_BREAKPOINT_OUT_OF_BYTECODE_BOUNDS**

vm**ASEBA_ASSERT_EMIT_BUFFER_TOO_LONG**

Variables

652, 735, 816, 896, 977, 1058, 1139, 1218, 1300, 1459, 1620, 1777, 1935, 2093, 2250, 2403, 2556, 2868, 3164, 3458, 3748, 4029, 4307, 4578, 4839, 5359, 5836, 6290, 6720, 7126, 7507, 7861, 8203, 8825, 9357, 9839, 10260, 10640, 10976, 11281, 11557, 12037, 12425, 12755, 13036, 13277, 13486, 13671, 13837, 14112, 14331, 14514, 14666, 14796, 14907, 15003, 15091, 15235, 15348, 15441, 15519, 15585, 15642, 15691, 15736, 15808, 15865, 15912, 15951, 15984, 16013, 16037, 16060, 16096, 16125, 16148, 16168, 16184, 16199, 16211, 16222, 16240, 16255, 16266, 16276, 16284, 16292, 16298, 16303, 16312, 16320, 16325, 16331, 16334, 16338, 16341, 16344, 16348, 16352, 16355, 16357, 16360, 16361, 16363, 16364, 16366, 16369, 16369, 16371, 16372, 16373, 16373, 16375, 16375, 16377, 16376, 16378, 16378, 16378, 16379, 16379, 16380, 16380, 16380, 16382, 16381, 16381, 16381, 16382, 16382, 16382, 16382, 16382, 16382, 16384, 16383, 16383, 16383, 16383, 16383, 16383, 16383, 16383, 16383, 16383, 16383, 16384, 16384, 16384, 16384, 16384, 16384, 16384, 16384, 16384, 16384, 16384, 16384, 16384, 16384}]
0, 403, 804, 1207, 1608, 2010, 2411, 2812, 3212, 3612, 4011, 4411, 4808, 5206, 5603, 5998, 6393, 6787, 7180, 7572, 7962, 8352, 8740, 9127, 9513, 9896, 10279, 10660, 11040, 11417, 11794, 12167, 12540, 12911, 13279, 13646, 14010, 14373, 14733, 15091, 15447, 15801, 16151, 16500, 16846, 17190, 17531, 17869, 18205, 18538, 18868, 19196, 19520, 19842, 20160, 20476, 20788, 21097, 21403, 21706, 22006, 22302, 22595, 22884, 23171, 23453, 23732, 24008, 24279, 24548, 24812, 25073, 25330, 25583, 25833, 26078, 26320, 26557, 26791, 27020, 27246, 27467, 27684, 27897, 28106, 28311, 28511, 28707, 28899, 29086, 29269, 29448, 29622, 29792, 29957, 30117, 30274, 30425, 30572, 30715, 30852, 30985, 31114, 31238, 31357, 31471, 31581, 31686, 31786, 31881, 31972, 32057, 32138, 32215, 32285, 32352, 32413, 32470, 32521, 32569, 32610, 32647, 32679, 32706, 32728, 32746, 32758, 32766, 32767, }]
“copies src to dest element by element”, {{-1, “dest”}, {-1, “src”}, {0, 0}}}]Description of AsebaNative_veccopy.
“fills dest with constant value”, {{-1, “dest”}, {1, “value”}, {0, 0}}}]Description of AsebaNative_vecfill.
“math.addscalar”, “add scalar to each element of src and store result to dest”, {{-1, “dest”}, {-1, “src”}, {1, “scalar”}, {0, 0}}}]Description of AsebaNative_vecaddscalar.
“math.add”, “adds src1 and src2, write to dest, element by element”, {{-1, “dest”}, {-1, “src1”}, {-1, “src2”}, {0, 0}}}]Description of AsebaNative_vecadd.
“math.sub”, “substracts src2 from src1 to dest, element by element”, {{-1, “dest”}, {-1, “src1”}, {-1, “src2”}, {0, 0}}}]Description of AsebaNative_vecsub.
“math.mul”, “multiplies src1 and src2 to dest, element by element”, {{-1, “dest”}, {-1, “src1”}, {-1, “src2”}, {0, 0}}}]Description of AsebaNative_vecadd.
“math.div”, “divides src1 by src2 to dest, element by element”, {{-1, “dest”}, {-1, “src1”}, {-1, “src2”}, {0, 0}}}]Description of AsebaNative_vecsub.
“math.min”, “writes the minimum of src1 and src2 to dest, element by element”, {{-1, “dest”}, {-1, “src1”}, {-1, “src2”}, {0, 0}}}]Description of AsebaNative_vecmin.
“math.max”, “writes the maximum of src1 and src2 to dest, element by element”, {{-1, “dest”}, {-1, “src1”}, {-1, “src2”}, {0, 0}}}]Description of AsebaNative_vecsmax.
“math.clamp”, “clamp src to low/high bounds into dest, element by element”, {{-1, “dest”}, {-1, “src”}, {-1, “low”}, {-1, “high”}, {0, 0}}}]Description of AsebaNative_vecclamp.

"math.dot", "writes the dot product of src1 and src2 to dest, after a shift", {{1, "dest"}, {-1, "src1"}, {-1, "src2"}, {1, "shift"}, {0, 0}}}]Description of AsebaNative_vecdot.
"math.stat", "performs statistics on src", {{-1, "src"}, {1, "min"}, {1, "max"}, {1, "mean"}, {0, 0}}}]Description of AsebaNative_vecstat.
"math.argbounds", "get the indices (argmin, argmax) of the limit values of src", {{-1, "src"}, {1, "argmin"}, {1, "argmax"}, {0, 0}}}]Description of AsebaNative_vecargbounds.
"sort array in place", {{-1, "array"}, {0, 0}}}]Description of AsebaNative_vecsort.
"math.muldiv", "performs dest = (a*b)/c in 32 bits element by element", {{-1, "dest"}, {-1, "a"}, {-1, "b"}, {-1, "c"}, {0, 0}}}]Description of AsebaNative_mathmuldiv.
"math.atan2", "performs dest = atan2(y,x) element by element", {{-1, "dest"}, {-1, "y"}, {-1, "x"}, {0, 0}}}]Description of AsebaNative_mathatan2.
"performs dest = sin(x) element by element", {{-1, "dest"}, {-1, "x"}, {0, 0}}}]Description of AsebaNative_mathsin.
"performs dest = cos(x) element by element", {{-1, "dest"}, {-1, "x"}, {0, 0}}}]Description of AsebaNative_mathcos.
"rotates v of angle a to dest", {{2, "dest"}, {2, "v"}, {1, "a"}, {0, 0}}}]Description of AsebaNative_mathrot2.
"performs dest = sqrt(x) element by element", {{-1, "dest"}, {-1, "x"}, {0, 0}}}]Description of AsebaNative_mathsqrt.
"math.nzseq", "write to dest the middle index of the largest sequence of non-zero elements from src, -1 if " "not found or if smaller than minLength", {{1, "dest"}, {-1, "src"}, {1, "minLength"}, {0, 0}}}]Description of AsebaNative_vecnonzerosequence.

uint16_t **rnd_state**
"fill array with random values", {{-1, "dest"}, {0, 0}}}]Description of AsebaNative_rand.
"report size of dest", {{-1, "deque"}, {1, "size"}, {0, 0}}}]Description of AsebaNative_deqsize.
"deque.get", "fill dest from deque at index", {{-1, "deque"}, {-2, "dest"}, {1, "index"}, {0, 0}}}]Description of AsebaNative_deqget.
"deque.set", "copies src to deque at index", {{-1, "deque"}, {-2, "src"}, {1, "index"}, {0, 0}}}]Description of AsebaNative_deqset.
"deque.insert", "shift deque at index by len(src) and insert src", {{-1, "deque"}, {-2, "src"}, {1, "index"}, {0, 0}}}]Description of AsebaNative_deqinsert.
"deque.erase", "shift deque at index by len to erase", {{-1, "deque"}, {1, "index"}, {1, "len"}, {0, 0}}}]Description of AsebaNative_deqerase.
"insert src before the front of deque", {{-1, "deque"}, {-2, "src"}, {0, 0}}}]Description of AsebaNative_deqpushfront.
"insert src after the back of deque", {{-1, "deque"}, {-2, "src"}, {0, 0}}}]Description of AsebaNative_deqpushback.
"deque.pop_front", "fill dest from front of deque then erase elements", {{-1, "deque"}, {-2, "dest"}, {0, 0}}}]Description of AsebaNative_deqpopfront.
"deque.pop_back", "fill dest from back of deque then erase elements", {{-1, "deque"}, {-2, "dest"}, {0, 0}}}]Description of AsebaNative_deqpopback.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_veccopy**
Description of AsebaNative_veccopy.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecfill**
Description of AsebaNative_vecfill.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecaddscalar**
Description of AsebaNative_vecaddscalar.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecadd**
Description of AsebaNative_vecadd.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecsub**
Description of AsebaNative_vecsub.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecmul**
Description of AsebaNative_vecadd.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecdiv**
Description of AsebaNative_vecsub.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecmin**
Description of AsebaNative_vecmin.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecmax**
Description of AsebaNative_vecsmax.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecclamp**
Description of AsebaNative_vecclamp.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecdot**
Description of AsebaNative_vecdot.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecstat**
Description of AsebaNative_vecstat.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecargbounds**
Description of AsebaNative_vecargbounds.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecsort**
Description of AsebaNative_vecsort.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_mathmuldiv**
Description of AsebaNative_mathmuldiv.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_mathatan2**
Description of AsebaNative_mathatan2.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_mathsin**
Description of AsebaNative_mathsin.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_mathcos**
Description of AsebaNative_mathcos.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_mathrot2**
Description of AsebaNative_mathrot2.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_mathsqrt**
Description of AsebaNative_mathsqrt.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_vecnonzerosequence**
Description of AsebaNative_vecnonzerosequence.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_rand**
Description of AsebaNative_rand.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_deqsize**
Description of AsebaNative_deqsize.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_deqget**
Description of AsebaNative_deqget.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_deqset**
Description of AsebaNative_deqset.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_deqinsert**
Description of AsebaNative_deqinsert.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_deqerase**
Description of AsebaNative_deqerase.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_deqpushfront**
Description of AsebaNative_deqpushfront.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_deqpushback**
Description of AsebaNative_deqpushback.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_deqpopfront**
Description of AsebaNative_deqpopfront.

const *AsebaNativeFunctionDescription* **AsebaNativeDescription_deqpopback**
Description of AsebaNative_deqpopback.

void const char* message

dir **aseba**

dir **compiler**

dir **vm**

page **index**

CHAPTER 9

Javascript API

class tymio.**Node**()

Node

tymio.Node.**disconnected**

Whether the node is disconnected

tymio.Node.**get_description**()

Get the description from the device The device must be in the ready state before requestibng the VM.

Throws **mobsya.fb.Error** –

Returns **external:Promise.<AsebaVMDescription>** –

tymio.Node.**id**

return the node id

tymio.Node.**lock**()

Lock the device Locking a device is akin to take sole ownership of it until the connection is closed or the unlock method is explicitely called

The device must be in the ready state before it can be locked. Once a device is locked, all client will see the device becoming busy.

If the device can not be locked, an {@link mobsya.fb.Error} is raised.

Throws **mobsya.fb.Error** –

tymio.Node.**run_aseba_program**()

Run the code currently loaded on the vm The device must be locked before calling this function

Throws **mobsya.fb.Error** –

tymio.Node.**send_aseba_program**(*code*)

Load an aseba program on the VM The device must be locked before calling this function

Arguments

- **code** (*external:String*) – the aseba code to load

Throws **mobsya.fb.Error** –

tymio.Node.**status**
The node status

tymio.Node.**status_str**
The node status converted to string.

tymio.Node.**unlock**()

Unlock the device Once a device is unlocked, all client will see the device becoming ready. Once unlock, a device can't be written to until loc

Throws **mobsya.fb.Error** –

class tymio.**Client**(*url*)

Arguments

- **url** (*external:String*) – : Web socket address

tymio.Client.**on_nodes_changed**(*nodes*)

Arguments

- **nodes** (*Array.<Node>*) – : Nodes whose status has changed.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Index

Symbols

A

B

D

E