daScript Standard Library

Release 0.2 beta

Anton Yudintsev

Mar 26, 2024

CONTENTS

1 Introduction 3

2 Built-in runtime 5

3 Math library 67

4 Math bit helpers 157

5 Boost package for math 165

6 File input output library 173

7 Random generator library 185

8 Network socket library 189

9 URI manipulation library based on UriParser 195

10 Boost package for the URI parser 205

11 Runtime type information library 209

12 AST manipulation library 255

13 Boost package for the AST 481

14 String manipulation library 507

15 Boost package for string manipulation library 529

16 Functional programming library 535

17 Jobs and threads 543

18 Boost package for jobs and threads 553

19 Cross-context evaluation helpers 561

20 JSON manipulation library 563

21 Boost package for JSON 569

22 Regular expression library 583

23 Boost package for REGEX 591

24 Documentation generator 593

25 Apply reflection pattern 595

26 Miscelanious algorithms 597

27 Miscelanious contract annotations 605

28 defer and defer_delete macros 607

29 if_not_null macro 609

30 instance_function function annotation 611

31 decltype macro and template function annotation 613

32 Template application helpers 615

33 Boost package for the miscelanious macro manipulations 633

34 is_local_xxx ast helpers 637

35 safe_addr macro 639

36 static_let macro 643

37 lpipe macro 645

38 Boost package for array manipulation 647

39 General prupose serialization 651

40 Loop unrolling 659

41 Assert once 661

42 DECS, AST block to loop 663

43 AST type ussage collection 665

44 Constant expression checker and substitution 667

45 Boost package for the builtin sort 669

46 Enumeration traits 671

47 C++ bindings generator 673

48 DECS, Daslang entity component system 675

49 Boost package for DECS 697

50 Coroutines and additional generator support 701

51 Interfaces 705

52 Export constructor 707

53 Faker 709

54 Fuzzer 721

55 Pattern matching 729

CHAPTER

ONE

INTRODUCTION

The Daslang standard libraries consist in a set of modules implemented in C++. While are not essential for the language, they provide a set of useful services that are commonly used by a wide range of applications (file I/O, regular expressions, etc.), plus they offer a foundation for developing additional libraries.

All libraries are implemented through the Daslang API and C++ runtime library. The modules are organized in the following way:

- *builtin runtime*
- *math basic mathematical routines*
- *fio - file input and output*
- *random - LCG random mathematical routines*
- *strings - string manipulation library*
- *daslib/strings_boost - boost package for STRINGS*
- *rtti - runtime type information and reflection library*
- *ast - compilation time information, reflection, and syntax tree library*
- *daslib/ast_boost - boost package for AST*
- *daslib/functional - high-order functions to support functional programming*
- *daslib/apply - apply reflection pattern*
- *daslib/json - JSON parser and writer*
- *daslib/json_boost - boost package for JSON*
- *daslib/regex - regular expression library*
- *daslib/regex_boost - boost package REGEX*
- *network - TCP raw socket server*
- *uriparser - URI manipulation library*
- *daslib/rst - RST documentation support*

CHAPTER

TWO

BUILT-IN RUNTIME

Builtin module is automatically required by any other das file. It includes basic language infrastructure, support for containers, heap, miscellaneous iterators, profiler, and interaction with host application.

2.1 Type aliases

print_flags is a bitfield

field	bit	value
escapeString	0	1
namesAndDimensions	1	2
typeQualifiers	2	4
refAddresses	3	8
humanReadable	4	16
singleLine	5	32

this bitfield specifies how exactly values are to be printed

2.2 Constants

DAS_MAX_FUNCTION_ARGUMENTS = 32

maximum number of arguments for the function. this is used to pre-allocate stack space for the function arguments

INT_MIN = -2147483648

minimum possible value of 'int'

INT_MAX = 2147483647

maximum possible value of 'int'

UINT_MAX = 0xffffffff

maximum possible value of 'uint'

LONG_MIN = -9223372036854775808

minimum possible value of 'int64'

LONG_MAX = 9223372036854775807

maximum possible value of 'int64'

ULONG_MAX = 0xffffffffffffffff

minimum possible value of 'uint64'

FLT_MIN = 1**.17549e-38f**

smallest possible non-zero value of 'float'. if u want minimum possible value use -*FLT_MAX*

FLT_MAX = 3**.40282e+38f**

maximum possible value of 'float'

DBL_MIN = 2**.22507e-308lf**

smallest possible non-zero value of 'double'. if u want minimum possible value use -*DBL_MAX*

DBL_MAX = 1**.79769e+308lf**

maximum possible value of 'double'

LOG_CRITICAL = 50000

indicates maximum log level. critial errors, panic, shutdown

LOG_ERROR = 40000

indicates log level recoverable errors

LOG_WARNING = 30000

indicates log level for API misuse, non-fatal errors

LOG_INFO = 20000

indicates log level for miscellaneous informative messages

LOG_DEBUG = 10000

indicates log level for debug messages

LOG_TRACE = 0

indicates log level for the most noisy debug and tracing messages

VEC_SEP = ","

Read-only string constant which is used to separate elements of vectors. By default its ","

print_flags_debugger = bitfield

printing flags similar to those used by the 'debug' function

2.3 Handled structures

HashBuilder

Helper structure to facilitate calculating hash values.

2.4 Function annotations

marker

marker annotation is used to attach arbitrary marker values to a function (in form of annotation arguments). its typically used for implementation of macros

generic

indicates that the function is generic, regardless of its argument types. generic functions will be instanced in the calling module

_macro

indicates that the function will be called during the macro pass, similar to *[init]*

macro_function

indicates that the function is part of the macro implementation, and will not be present in the final compiled context, unless explicitly called.

hint

Hints the compiler to use specific optimization.

jit

Explicitly marks (forces) function to be compiled with JIT compiler.

deprecated

deprecated annotation is used to mark a function as deprecated. it will generate a warning during compilation, and will not be callable from the final compiled context

alias_cmres

indicates that function always aliases cmres (copy or move result), and cmres optimizations are disabled.

never_alias_cmres

indicates that function never aliases cmres (copy or move result), and cmres checks will not be performed

export

indicates that function is to be exported to the final compiled context

pinvoke

indicates that the function is a pinvoke function, and will be called via pinvoke machinery

no_lint

indicates that the lint pass should be skipped for the specific function

sideeffects

indicates that the function should be treated as if it has side-effects. for example it will not be optimized out

run

ensures that the function is always evaluated at compilation time

unsafe_operation

indicates that function is unsafe, and will require *unsafe* keyword to be called

unsafe_outside_of_for

Marks function as unsafe to be called outside of the sources *for* loop.

no_aot

indicates that the AOT will not be generated for this specific function

init

indicates that the function would be called at the context initialization time

finalize

indicates that the function would be called at the context shutdown time

hybrid

indicates that the function is likely candidate for later patching, and the AOT will generate hybrid calls to it - instead of direct calls. that way modyfing the function will not affect AOT of other functions.

unsafe_deref

optimization, which indicates that pointer dereference, array and string indexing, and few other operations would not check for null or bounds

skip_lock_check

optimization, which indicates that lock checks are not needed in this function.

unused_argument

marks function arguments, which are unused. that way when code policies make unused arguments an error, a workaround can be provided

local_only

indicates that function can only accept local *make* expressions, like [[make tuple]] and [[make structure]]

expect_any_vector

indicates that function can only accept das::vector templates

builtin_array_sort

indicates sort function for builtin 'sort' machinery. used internally

2.5 Call macros

concept_assert

similar to regular *assert* function, but always happens at compilation time. it would also display the error message from where the asserted function was called from, not the assert line itself.

__builtin_table_set_insert

part of internal implementation for *insert* of the sets (tables with keys only).

__builtin_table_key_exists

part of internal implementation for *key_exists*

static_assert

similar to regular *assert* function, but always happens at compilation time

verify

assert for the expression with side effects. expression will not be optimized out if asserts are disabled

debug

prints value and returns that same value

assert

throws panic if first operand is false. can be disabled. second operand is error message

memzero

initializes section of memory with '0'

__builtin_table_find

part of internal implementation for *find*

invoke

invokes block, function, or lambda

__builtin_table_erase

part of internal implementation for *erase*

2.6 Reader macros

_esc

returns raw string input, without regards for escape sequences. For example %_esc\n\r%_esc will return 4 character string '','n',' '','r'

2.7 Typeinfo macros

rtti_classinfo

Generates TypeInfo for the class initialization.

2.8 Handled types

das_string

das::string which is typically std::string or equivalent

clock

das::Time which is a wrapper around *time_t*

2.9 Structure macros

comment

[comment] macro, which does absolutely nothing but holds arguments.

macro_interface

[macro_interface] specifies that class and its inherited children are used as a macro interfaces, and would not be exported by default.

skip_field_lock_check

optimization, which indicates that the structure does not need lock checks.

cpp_layout

[cpp_layout] specifies that structure uses C++ memory layout rules, as oppose to native Daslang memory layout rules.

persistent

[persistent] annotation specifies that structure is allocated (via new) on the C++ heap, as oppose to Daslang context heap.

2.10 Containers

- *clear (array:array implicit;context:__context const;at:__lineInfo const) : void*
- *length (array:array const implicit) : int*
- *capacity (array:array const implicit) : int*
- *empty (iterator:iterator const implicit) : bool*
- *length (table:table const implicit) : int*
- *capacity (table:table const implicit) : int*
- *empty (str:string const implicit) : bool*
- *empty (str:$::das_string const implicit) : bool*
- *resize (Arr:array<auto(numT)> -const;newSize:int const) : auto*
- *resize_no_init (Arr:array<auto(numT)> -const;newSize:int const) : auto*
- *reserve (Arr:array<auto(numT)> -const;newSize:int const) : auto*
- *pop (Arr:array<auto(numT)> -const) : auto*
- *push (Arr:array<auto(numT)> -const;value:numT const -#;at:int const) : auto*
- *push (Arr:array<auto(numT)> -const;value:numT const -#) : auto*
- *push (Arr:array<auto(numT)> -const;varr:array<numT> const -#) : auto*
- *push (Arr:array<auto(numT)> -const;varr:numT const[] -#) : auto*
- *push (Arr:array<auto(numT)[]> -const;varr:numT const[] -#) : auto*
- *emplace (Arr:array<auto(numT)> -const;value:numT& -const -#;at:int const) : auto*
- *emplace (Arr:array<auto(numT)> -const;value:numT& -const -#) : auto*
- *emplace (Arr:array<auto(numT)> -const;value:numT[] -const -#) : auto*

- *emplace (Arr:array<auto(numT)[]> -const;value:numT[] -const -#) : auto*
- *push_clone (Arr:array<auto(numT)> -const;value:numT const|numT const# const;at:int const) : auto*
- *push_clone (Arr:array<auto(numT)> -const;value:numT const|numT const# const) : auto*
- *push_clone (Arr:array<auto(numT)> -const;varr:numT const[]) : auto*
- *push_clone (Arr:array<auto(numT)[]> -const;varr:numT const[]) : auto*
- *push_clone (A:auto(CT) -const -#;b:auto(TT) const|auto(TT) const# const) : auto*
- *back (a:array<auto(TT)> ==const -const) : TT&*
- *back (a:array<auto(TT)># ==const -const) : TT&#*
- *back (a:array<auto(TT)> const ==const) : TT const&*
- *back (a:array<auto(TT)> const# ==const) : TT const&#*
- *back (arr:auto(TT) ==const -const) : auto&*
- *back (arr:auto(TT) const ==const) : auto const&*
- *erase (Arr:array<auto(numT)> -const;at:int const) : auto*
- *erase (Arr:array<auto(numT)> -const;at:int const;count:int const) : auto*
- *length (a:auto const[]) : int*
- *empty (a:array<auto> const|array<auto> const# const) : bool*
- *empty (a:table<auto;auto> const|table<auto;auto> const# const) : bool*
- *find (Tab:table<auto(keyT);auto(valT)> const|table<auto(keyT);auto(valT)> const# const;at:keyT const -#;blk:block<(p:valT? const#):void> const) : auto*
- *find (Tab:table<auto(keyT);void> const;at:keyT const|keyT const# const;blk:block<(p:void? const):void> const) : auto*
- *get (Tab:table<auto(keyT);auto(valT)> const# ==const;at:keyT const -#;blk:block<(p:valT const&#):void> const) : auto*
- *get (Tab:table<auto(keyT);auto(valT)> const ==const;at:keyT const -#;blk:block<(p:valT const&):void> const) : auto*
- *get (Tab:table<auto(keyT);auto(valT)># ==const -const;at:keyT const -#;blk:block<(var p:valT&# -const):void> const) : auto*
- *get (Tab:table<auto(keyT);auto(valT)> ==const -const;at:keyT const -#;blk:block<(var p:valT& -const):void> const) : auto*
- *get (Tab:table<auto(keyT);void> const;at:keyT const|keyT const# const;blk:block<(var p:void? -const):void> const) : auto*
- *find_if_exists (Tab:table<auto(keyT);auto(valT)> const;at:keyT const -#;blk:block<(p:valT const&):void> const) : auto*
- *find_if_exists (Tab:table<auto(keyT);auto(valT)> const#;at:keyT const -#;blk:block<(p:valT const&#):void> const) : auto*
- *find_if_exists (Tab:table<auto(keyT);void> const;at:keyT const -#;blk:block<(p:void? const):void> const) : auto*
- *find_for_edit (Tab:table<auto(keyT);auto(valT)> -const;at:keyT const -#;blk:block<(var p:valT?# -const):void> const) : auto*

- find_for_edit (Tab:table<auto(keyT);void> -const;at:keyT const|keyT const# const;blk:block<(var p:void? -const):void> const) : auto
- find_for_edit (Tab:table<auto(keyT);auto(valT)> -const|table<auto(keyT);auto(valT)># -const -const;at:keyT const -#) : valT?
- find_for_edit (Tab:table<auto(keyT);void> -const;at:keyT const|keyT const# const) : void?
- find_for_edit_if_exists (Tab:table<auto(keyT);auto(valT)># -const;at:keyT const -#;blk:block<(var p:valT&# -const):void> const) : auto
- find_for_edit_if_exists (Tab:table<auto(keyT);auto(valT)> -const;at:keyT const -#;blk:block<(var p:valT& -const):void> const) : auto
- find_for_edit_if_exists (Tab:table<auto(keyT);void> -const;at:keyT const|keyT const# const;blk:block<(var p:void? -const):void> const) : auto
- erase (Tab:table<auto(keyT);auto(valT)> -const;at:string const#) : bool
- erase (Tab:table<auto(keyT);auto(valT)> -const;at:keyT const|keyT const# const) : bool
- insert (Tab:table<auto(keyT);void> -const;at:keyT const|keyT const# const) : auto
- key_exists (Tab:table<auto(keyT);auto(valT)> const|table<auto(keyT);auto(valT)> const# const;at:string const#) : bool
- key_exists (Tab:table<auto(keyT);auto(valT)> const|table<auto(keyT);auto(valT)> const# const;at:keyT const|keyT const# const) : bool
- copy_to_local (a:auto(TT) const) : TT -const
- move_to_local (a:auto(TT)& -const) : TT -const -&
- keys (a:table<auto(keyT);auto(valT)> const|table<auto(keyT);auto(valT)> const# const) : iterator<keyT const&>
- values (a:table<auto(keyT);void> const ==const|table<auto(keyT);void> const# ==const const) : auto
- values (a:table<auto(keyT);void> ==const -const|table<auto(keyT);void># ==const -const -const) : auto
- values (a:table<auto(keyT);auto(valT)> const ==const|table<auto(keyT);auto(valT)> const# ==const const) : iterator<valT const&>
- values (a:table<auto(keyT);auto(valT)> ==const -const|table<auto(keyT);auto(valT)># ==const -const -const) : iterator<valT&>
- lock (Tab:table<auto(keyT);auto(valT)> const|table<auto(keyT);auto(valT)> const# const;blk:block<(t:table<keyT;valT> const#):void> const) : auto
- lock_forever (Tab:table<auto(keyT);auto(valT)> -const|table<auto(keyT);auto(valT)># -const -const) : table<keyT;valT>#
- next (it:iterator<auto(TT)> const;value:TT& -const) : bool
- each (rng:range const) : iterator<int>
- each (str:string const) : iterator<int>
- each (a:auto(TT) const[]) : iterator<TT&>
- each (a:array<auto(TT)> const) : iterator<TT&>
- each (a:array<auto(TT)> const#) : iterator<TT&#>
- each (lam:lambda<(var arg:auto(argT) -const):bool> const) : iterator<argT -&>
- each_ref (lam:lambda<(var arg:auto(argT)? -const):bool> const) : iterator<argT&>

- each_enum (tt:auto(TT) const) : iterator<TT -const -&>
- nothing (it:iterator<auto(TT)> -const) : iterator<TT>
- to_array (it:iterator<auto(TT)> const) : array<TT -const -&>
- to_array (a:auto(TT) const[]) : array<TT -const>
- to_array_move (a:auto(TT)[] -const) : array<TT -const>
- to_array_move (a:auto(TT) -const) : array<TT -const>
- to_table (a:tuple<auto(keyT);auto(valT)> const[]) : table<keyT -const;valT>
- to_table (a:auto(keyT) const[]) : table<keyT -const;void>
- to_table_move (a:auto(keyT)[] -const) : table<keyT -const;void>
- to_table_move (a:tuple<auto(keyT);auto(valT)>[] -const) : table<keyT -const;valT>
- sort (a:auto(TT)[] -const|auto(TT)[]# -const -const) : auto
- sort (a:array<auto(TT)> -const|array<auto(TT)># -const -const) : auto
- sort (a:auto(TT)[] -const|auto(TT)[]# -const -const;cmp:block<(x:TT const;y:TT const):bool> const) : auto
- sort (a:array<auto(TT)> -const|array<auto(TT)># -const -const;cmp:block<(x:TT const;y:TT const):bool> const) : auto
- lock (a:array<auto(TT)> ==const -const|array<auto(TT)># ==const -const -const;blk:block<(var x:array<TT># -const):auto> const) : auto
- lock (a:array<auto(TT)> const ==const|array<auto(TT)> const# ==const const;blk:block<(x:array<TT> const#):auto> const) : auto
- find_index (arr:array<auto(TT)> const|array<auto(TT)> const# const;key:TT const) : auto
- find_index (arr:auto(TT) const[]|auto(TT) const[]# const;key:TT const) : auto
- find_index (arr:iterator<auto(TT)> const;key:TT const -&) : auto
- find_index_if (arr:array<auto(TT)> const|array<auto(TT)> const# const;blk:block<(key:TT const):bool> const) : auto
- find_index_if (arr:auto(TT) const[]|auto(TT) const[]# const;blk:block<(key:TT const):bool> const) : auto
- find_index_if (arr:iterator<auto(TT)> const;blk:block<(key:TT const -&):bool> const) : auto
- has_value (a:auto const;key:auto const) : auto
- subarray (a:auto(TT) const[];r:range const) : auto
- subarray (a:auto(TT) const[];r:urange const) : auto
- subarray (a:array<auto(TT)> const;r:range const) : auto
- subarray (a:array<auto(TT)> const;r:urange const) : auto
- move_to_ref (a:auto& -const;b:auto -const) : auto
- clear (t:table<auto(KT);auto(VT)> -const) : auto

clear (*array: array implicit*)

argument	argument type
array	array implicit

clear will clear whole table or array *arg*. The size of *arg* after clear is 0.

length (*array: array const implicit*)

length returns int

argument	argument type
array	array const implicit

length will return current size of table or array *arg*.

capacity (*array: array const implicit*)

capacity returns int

argument	argument type
array	array const implicit

capacity will return current capacity of table or array *arg*. Capacity is the count of elements, allocating (or pushing) until that size won't cause reallocating dynamic heap.

empty (*iterator: iterator const implicit*)

empty returns bool

argument	argument type
iterator	iterator const implicit

returns true if iterator is empty, i.e. would not produce any more values or uninitialized

length (*table: table const implicit*)

length returns int

argument	argument type
table	table const implicit

length will return current size of table or array *arg*.

capacity (*table: table const implicit*)

capacity returns int

argument	argument type
table	table const implicit

capacity will return current capacity of table or array *arg*. Capacity is the count of elements, allocating (or pushing) until that size won't cause reallocating dynamic heap.

empty (*str: string const implicit*)

empty returns bool

argument	argument type
str	string const implicit

returns true if iterator is empty, i.e. would not produce any more values or uninitialized

empty (*str: das_string const implicit*)

empty returns bool

argument	argument type
str	*builtin::das_string* const implicit

returns true if iterator is empty, i.e. would not produce any more values or uninitialized

resize (*Arr: array<auto(numT)>; newSize: int const*)

resize returns auto

argument	argument type
Arr	array<auto(numT)>
newSize	int const

Resize will resize *array_arg* array to a new size of *new_size*. If new_size is bigger than current, new elements will be zeroed.

resize_no_init (*Arr: array<auto(numT)>; newSize: int const*)

resize_no_init returns auto

argument	argument type
Arr	array<auto(numT)>
newSize	int const

Resize will resize *array_arg* array to a new size of *new_size*. If new_size is bigger than current, new elements will be left uninitialized.

reserve (*Arr: array<auto(numT)>; newSize: int const*) reserve returns auto

argument	argument type
Arr	array<auto(numT)>
newSize	int const

makes sure array has sufficient amount of memory to hold specified number of elements. reserving arrays will speed up pushing to it

`pop` (*Arr: array<auto(numT)>*)

pop returns auto

argument	argument type
Arr	array<auto(numT)>

removes last element of the array

`push` (*Arr: array<auto(numT)>; value: numT const; at: int const*)

push returns auto

argument	argument type
Arr	array<auto(numT)>
value	numT const
at	int const

push will push to dynamic array *array_arg* the content of *value*. *value* has to be of the same type (or const reference to same type) as array values. if *at* is provided *value* will be pushed at index *at*, otherwise to the end of array. The *content* of value will be copied (assigned) to it.

`push` (*Arr: array<auto(numT)>; value: numT const*)

push returns auto

argument	argument type
Arr	array<auto(numT)>
value	numT const

push will push to dynamic array *array_arg* the content of *value*. *value* has to be of the same type (or const reference to same type) as array values. if *at* is provided *value* will be pushed at index *at*, otherwise to the end of array. The *content* of value will be copied (assigned) to it.

push (*Arr: array<auto(numT)>; varr: array<numT> const*)

push returns auto

argument	argument type
Arr	array<auto(numT)>
varr	array<numT> const

push will push to dynamic array *array_arg* the content of *value*. *value* has to be of the same type (or const reference to same type) as array values. if *at* is provided *value* will be pushed at index *at*, otherwise to the end of array. The *content* of value will be copied (assigned) to it.

push (*Arr: array<auto(numT)>; varr: numT const[]*)

push returns auto

argument	argument type
Arr	array<auto(numT)>
varr	numT const[-1]

push will push to dynamic array *array_arg* the content of *value*. *value* has to be of the same type (or const reference to same type) as array values. if *at* is provided *value* will be pushed at index *at*, otherwise to the end of array. The *content* of value will be copied (assigned) to it.

push (*Arr: array<auto(numT)[]>; varr: numT const[]*)

push returns auto

argument	argument type
Arr	array<auto(numT)[-1]>
varr	numT const[-1]

push will push to dynamic array *array_arg* the content of *value*. *value* has to be of the same type (or const reference to same type) as array values. if *at* is provided *value* will be pushed at index *at*, otherwise to the end of array. The *content* of value will be copied (assigned) to it.

emplace (*Arr: array<auto(numT)>; value: numT&; at: int const*)

emplace returns auto

argument	argument type
Arr	array<auto(numT)>
value	numT&
at	int const

emplace will push to dynamic array *array_arg* the content of *value*. *value* has to be of the same type (or const reference to same type) as array values. if *at* is provided *value* will be pushed at index *at*, otherwise to the end of array. The *content* of value will be moved (<-) to it.

emplace (*Arr: array<auto(numT)>; value: numT&*)

emplace returns auto

argument	argument type
Arr	array<auto(numT)>
value	numT&

emplace will push to dynamic array *array_arg* the content of *value*. *value* has to be of the same type (or const reference to same type) as array values. if *at* is provided *value* will be pushed at index *at*, otherwise to the end of array. The *content* of value will be moved (<-) to it.

emplace (*Arr: array<auto(numT)>; value: numT[]*)

emplace returns auto

argument	argument type
Arr	array<auto(numT)>
value	numT[-1]

emplace will push to dynamic array *array_arg* the content of *value*. *value* has to be of the same type (or const reference to same type) as array values. if *at* is provided *value* will be pushed at index *at*, otherwise to the end of array. The *content* of value will be moved (<-) to it.

emplace (*Arr: array<auto(numT)[]>; value: numT[]*)

emplace returns auto

argument	argument type
Arr	array<auto(numT)[-1]>
value	numT[-1]

emplace will push to dynamic array *array_arg* the content of *value*. *value* has to be of the same type (or const reference to same type) as array values. if *at* is provided *value* will be pushed at index *at*, otherwise to the end of array. The *content* of value will be moved (<-) to it.

push_clone (*Arr: array<auto(numT)>; value: numT const|numT const# const; at: int const*)

push_clone returns auto

argument	argument type
Arr	array<auto(numT)>
value	option const
at	int const

similar to *push*, only values would be cloned instead of copied

push_clone (*Arr: array<auto(numT)>; value: numT const|numT const# const*)

push_clone returns auto

argument	argument type
Arr	array<auto(numT)>
value	option const

similar to *push*, only values would be cloned instead of copied

push_clone (*Arr: array<auto(numT)>; varr: numT const[]*)

push_clone returns auto

argument	argument type
Arr	array<auto(numT)>
varr	numT const[-1]

similar to *push*, only values would be cloned instead of copied

push_clone (*Arr: array<auto(numT)[]>; varr: numT const[]*)

push_clone returns auto

argument	argument type
Arr	array<auto(numT)[-1]>
varr	numT const[-1]

similar to *push*, only values would be cloned instead of copied

push_clone (*A: auto(CT); b: auto(TT) const|auto(TT) const# const*)

push_clone returns auto

argument	argument type
A	auto(CT)
b	option const

similar to *push*, only values would be cloned instead of copied

back (*a: array<auto(TT)> ==const*)

back returns TT&

argument	argument type
a	array<auto(TT)>!

returns last element of the array

back (*a: array<auto(TT)># ==const*)

back returns TT&#

argument	argument type
a	array<auto(TT)>#!

returns last element of the array

back (*a: array<auto(TT)> const ==const*)

back returns TT const&

argument	argument type
a	array<auto(TT)> const!

returns last element of the array

back (*a: array<auto(TT)> const# ==const*)

back returns TT const&#

argument	argument type
a	array<auto(TT)> const#!

returns last element of the array

back (*arr: auto(TT) ==const*)

back returns auto&

argument	argument type
arr	auto(TT)!

returns last element of the array

back (*arr: auto(TT) const ==const*)

back returns auto const&

argument	argument type
arr	auto(TT) const!

returns last element of the array

erase (*Arr: array<auto(numT)>; at: int const*)

erase returns auto

argument	argument type
Arr	array<auto(numT)>
at	int const

erase will erase *at* index element in *arg* array.

erase (*Arr: array<auto(numT)>; at: int const; count: int const*)

erase returns auto

argument	argument type
Arr	array<auto(numT)>
at	int const
count	int const

erase will erase *at* index element in *arg* array.

length (*a: auto const[]*)

length returns int

argument	argument type
a	auto const[-1]

length will return current size of table or array *arg*.

empty (*a: array<auto> const|array<auto> const# const*)

empty returns bool

argument	argument type
a	option const

returns true if iterator is empty, i.e. would not produce any more values or uninitialized

empty (*a: table<auto;auto> const|table<auto;auto> const# const*)

empty returns bool

argument	argument type
a	option const

returns true if iterator is empty, i.e. would not produce any more values or uninitialized

find (*Tab: table<auto(keyT);auto(valT)> const|table<auto(keyT);auto(valT)> const# const; at: keyT const; blk: block<(p:valT? const#):void> const*)

find returns auto

> **Warning:** This function is deprecated.

argument	argument type
Tab	option const
at	keyT const
blk	block<(p:valT? const#):void> const

will execute *block_arg* with argument pointer-to-value in *table_arg* pointing to value indexed by *key*, or null if *key* doesn't exist in *table_arg*.

find (*Tab: table<auto(keyT);void> const; at: keyT const|keyT const# const; blk: block<(p:void? const):void> const*)

find returns auto

Warning: This function is deprecated.

argument	argument type
Tab	table<auto(keyT);void> const
at	option const
blk	block<(p:void? const):void> const

will execute *block_arg* with argument pointer-to-value in *table_arg* pointing to value indexed by *key*, or null if *key* doesn't exist in *table_arg*.

get (*Tab: table<auto(keyT);auto(valT)> const# ==const; at: keyT const; blk: block<(p:valT const&#):void> const*)

get returns auto

argument	argument type
Tab	table<auto(keyT);auto(valT)> const#!
at	keyT const
blk	block<(p:valT const&#):void> const

will execute *block_arg* with argument reference-to-value in *table_arg* referencing value indexed by *key*. Will return false if *key* doesn't exist in *table_arg*, otherwise true.

get (*Tab: table<auto(keyT);auto(valT)> const ==const; at: keyT const; blk: block<(p:valT const&):void> const*)

get returns auto

argument	argument type
Tab	table<auto(keyT);auto(valT)> const!
at	keyT const
blk	block<(p:valT const&):void> const

will execute *block_arg* with argument reference-to-value in *table_arg* referencing value indexed by *key*. Will return false if *key* doesn't exist in *table_arg*, otherwise true.

get (*Tab: table<auto(keyT);auto(valT)># ==const; at: keyT const; blk: block<(var p:valT&# -const):void> const*)

get returns auto

argument	argument type
Tab	table<auto(keyT);auto(valT)>#!
at	keyT const
blk	block<(p:valT&#):void> const

will execute *block_arg* with argument reference-to-value in *table_arg* referencing value indexed by *key*. Will return false if *key* doesn't exist in *table_arg*, otherwise true.

get (*Tab: table<auto(keyT);auto(valT)> ==const; at: keyT const; blk: block<(var p:valT& -const):void> const*)

get returns auto

argument	argument type
Tab	table<auto(keyT);auto(valT)>!
at	keyT const
blk	block<(p:valT&):void> const

will execute *block_arg* with argument reference-to-value in *table_arg* referencing value indexed by *key*. Will return false if *key* doesn't exist in *table_arg*, otherwise true.

get (*Tab: table<auto(keyT);void> const; at: keyT const|keyT const# const; blk: block<(var p:void? -const):void> const*)

get returns auto

argument	argument type
Tab	table<auto(keyT);void> const
at	option const
blk	block<(p:void?):void> const

will execute *block_arg* with argument reference-to-value in *table_arg* referencing value indexed by *key*. Will return false if *key* doesn't exist in *table_arg*, otherwise true.

find_if_exists (*Tab: table<auto(keyT);auto(valT)> const; at: keyT const; blk: block<(p:valT const&):void> const*)

find_if_exists returns auto

Warning: This function is deprecated.

argument	argument type
Tab	table<auto(keyT);auto(valT)> const
at	keyT const
blk	block<(p:valT const&):void> const

similar to find, but the block will only be called, if the key is found

find_if_exists (*Tab: table<auto(keyT);auto(valT)> const#; at: keyT const; blk: block<(p:valT const&#):void> const*)

find_if_exists returns auto

> **Warning:** This function is deprecated.

argument	argument type
Tab	table<auto(keyT);auto(valT)> const#
at	keyT const
blk	block<(p:valT const&#):void> const

similar to find, but the block will only be called, if the key is found

find_if_exists (*Tab: table<auto(keyT);void> const; at: keyT const; blk: block<(p:void? const):void> const*)

find_if_exists returns auto

> **Warning:** This function is deprecated.

argument	argument type
Tab	table<auto(keyT);void> const
at	keyT const
blk	block<(p:void? const):void> const

similar to find, but the block will only be called, if the key is found

find_for_edit (*Tab: table<auto(keyT);auto(valT)>; at: keyT const; blk: block<(var p:valT?# -const):void> const*)

find_for_edit returns auto

> **Warning:** This function is deprecated.

argument	argument type
Tab	table<auto(keyT);auto(valT)>
at	keyT const
blk	block<(p:valT?#):void> const

similar to find, but pointer to the value would be read-write

find_for_edit (*Tab: table<auto(keyT);void>; at: keyT const|keyT const# const; blk: block<(var p:void? -const):void> const*)

find_for_edit returns auto

> **Warning:** This function is deprecated.

argument	argument type
Tab	table<auto(keyT);void>
at	option const
blk	block<(p:void?):void> const

similar to find, but pointer to the value would be read-write

find_for_edit (*Tab: table<auto(keyT);auto(valT)> -const|table<auto(keyT);auto(valT)># -const; at: keyT const*)

find_for_edit returns valT?

> **Warning:** This is unsafe operation.

> **Warning:** This function is deprecated.

argument	argument type
Tab	option
at	keyT const

similar to find, but pointer to the value would be read-write

find_for_edit (*Tab: table<auto(keyT);void>; at: keyT const|keyT const# const*)

find_for_edit returns void?

> **Warning:** This is unsafe operation.

> **Warning:** This function is deprecated.

argument	argument type
Tab	table<auto(keyT);void>
at	option const

similar to find, but pointer to the value would be read-write

find_for_edit_if_exists (*Tab: table<auto(keyT);auto(valT)>#; at: keyT const; blk: block<(var p:valT&# -const):void> const*)

find_for_edit_if_exists returns auto

> **Warning:** This function is deprecated.

argument	argument type
Tab	table<auto(keyT);auto(valT)>#
at	keyT const
blk	block<(p:valT&#):void> const

similar to find_if_exists, but pointer to the value would be read-write

find_for_edit_if_exists (*Tab: table<auto(keyT);auto(valT)>; at: keyT const; blk: block<(var p:valT& -const):void> const*)

find_for_edit_if_exists returns auto

> **Warning:** This function is deprecated.

argument	argument type
Tab	table<auto(keyT);auto(valT)>
at	keyT const
blk	block<(p:valT&):void> const

similar to find_if_exists, but pointer to the value would be read-write

find_for_edit_if_exists (*Tab: table<auto(keyT);void>; at: keyT const|keyT const# const; blk: block<(var p:void? -const):void> const*)

find_for_edit_if_exists returns auto

> **Warning:** This function is deprecated.

argument	argument type
Tab	table<auto(keyT);void>
at	option const
blk	block<(p:void?):void> const

similar to find_if_exists, but pointer to the value would be read-write

erase (*Tab: table<auto(keyT);auto(valT)>; at: string const#*)

erase returns bool

argument	argument type
Tab	table<auto(keyT);auto(valT)>
at	string const#

erase will erase *at* index element in *arg* array.

erase (*Tab: table<auto(keyT);auto(valT)>; at: keyT const|keyT const# const*)

erase returns bool

argument	argument type
Tab	table<auto(keyT);auto(valT)>
at	option const

erase will erase *at* index element in *arg* array.

insert (*Tab: table<auto(keyT);void>; at: keyT const|keyT const# const*)

insert returns auto

argument	argument type
Tab	table<auto(keyT);void>
at	option const

inserts key into the set (table with no values) *Tab*

key_exists (*Tab: table<auto(keyT);auto(valT)> const|table<auto(keyT);auto(valT)> const# const; at: string const#*)

key_exists returns bool

argument	argument type
Tab	option const
at	string const#

will return true if element *key* exists in table *table_arg*.

key_exists (*Tab: table<auto(keyT);auto(valT)> const|table<auto(keyT);auto(valT)> const# const; at: keyT const|keyT const# const*)

key_exists returns bool

argument	argument type
Tab	option const
at	option const

will return true if element *key* exists in table *table_arg*.

copy_to_local (*a: auto(TT) const*)

copy_to_local returns TT

argument	argument type
a	auto(TT) const

copies value and returns it as local value on stack. used to work around aliasing issues

move_to_local (*a: auto(TT)&*)

move_to_local returns TT

argument	argument type
a	auto(TT)&

moves value and returns it as local value on stack. used to work around aliasing issues

keys (*a: table<auto(keyT);auto(valT)> const|table<auto(keyT);auto(valT)> const# const*)

keys returns iterator<keyT const&>

argument	argument type
a	option const

returns iterator to all keys of the table

values (*a: table<auto(keyT);void> const ==const|table<auto(keyT);void> const# ==const const*)

values returns auto

argument	argument type
a	option const

returns iterator to all values of the table

values (*a: table<auto(keyT);void> ==const -const|table<auto(keyT);void># ==const -const*)

values returns auto

argument	argument type
a	option

returns iterator to all values of the table

values (*a: table<auto(keyT);auto(valT)> const ==const|table<auto(keyT);auto(valT)> const# ==const const*)

values returns iterator<valT const&>

argument	argument type
a	option const

returns iterator to all values of the table

values (*a: table<auto(keyT);auto(valT)> ==const -const|table<auto(keyT);auto(valT)># ==const -const*) values returns iterator<valT&>

argument	argument type
a	option

returns iterator to all values of the table

lock (*Tab: table<auto(keyT);auto(valT)> const|table<auto(keyT);auto(valT)> const# const; blk: block<(t:table<keyT;valT> const#):void> const*)

lock returns auto

argument	argument type
Tab	option const
blk	block<(t:table<keyT;valT> const#):void> const

locks array or table for the duration of the block invocation, so that it can’t be resized. values can’t be pushed or popped, etc.

lock_forever (*Tab: table<auto(keyT);auto(valT)> -const|table<auto(keyT);auto(valT)># -const*)

lock_forever returns table<keyT;valT>#

argument	argument type
Tab	option

locks array or table forever

next (*it: iterator<auto(TT)> const; value: TT&*)

next returns bool

argument	argument type
it	iterator<auto(TT)> const
value	TT&

returns next element in the iterator as the ‘value’. result is true if there is element returned, or false if iterator is null or empty

each (*rng: range const*)

each returns iterator<int>

argument	argument type
rng	range const

returns iterator, which iterates though each element of the object. object can be range, static or dynamic array, another iterator.

each (*str: string const*)

each returns iterator<int>

argument	argument type
str	string const

returns iterator, which iterates though each element of the object. object can be range, static or dynamic array, another iterator.

each (*a: auto(TT) const[]*)

each returns iterator<TT&>

argument	argument type
a	auto(TT) const[-1]

returns iterator, which iterates though each element of the object. object can be range, static or dynamic array, another iterator.

each (*a: array<auto(TT)> const*)

each returns iterator<TT&>

argument	argument type
a	array<auto(TT)> const

returns iterator, which iterates though each element of the object. object can be range, static or dynamic array, another iterator.

each (*a: array<auto(TT)> const#*)

each returns iterator<TT&#>

argument	argument type
a	array<auto(TT)> const#

returns iterator, which iterates though each element of the object. object can be range, static or dynamic array, another iterator.

each (*lam: lambda<(var arg:auto(argT) -const):bool> const*)

each returns iterator<argT>

argument	argument type
lam	lambda<(arg:auto(argT)):bool> const

returns iterator, which iterates though each element of the object. object can be range, static or dynamic array, another iterator.

each_ref (*lam: lambda<(var arg:auto(argT)? -const):bool> const*)

each_ref returns iterator<argT&>

argument	argument type
lam	lambda<(arg:auto(argT)?):bool> const

similar to each, but iterator returns references instead of values

each_enum (*tt: auto(TT) const*)

each_enum returns iterator<TT>

argument	argument type
tt	auto(TT) const

iterates over each element in the enumeration

nothing (*it: iterator<auto(TT)>*)

nothing returns iterator<TT>

argument	argument type
it	iterator<auto(TT)>

returns empty iterator

to_array (*it: iterator<auto(TT)> const*)

to_array returns array<TT>

argument	argument type
it	iterator<auto(TT)> const

will convert argument (static array, iterator, another dynamic array) to an array. argument elements will be cloned

to_array (*a: auto(TT) const[]*)

to_array returns array<TT>

argument	argument type
a	auto(TT) const[-1]

will convert argument (static array, iterator, another dynamic array) to an array. argument elements will be cloned

to_array_move (*a: auto(TT)[]*)

to_array_move returns array<TT>

argument	argument type
a	auto(TT)[-1]

will convert argument (static array, iterator, another dynamic array) to an array. argument elements will be copied or moved

to_array_move (*a: auto(TT)*)

to_array_move returns array<TT>

argument	argument type
a	auto(TT)

will convert argument (static array, iterator, another dynamic array) to an array. argument elements will be copied or moved

to_table (*a: tuple<auto(keyT);auto(valT)> const[]*)

to_table returns table<keyT;valT>

argument	argument type
a	tuple<auto(keyT);auto(valT)> const[-1]

will convert an array of key-value tuples into a table<key;value> type. arguments will be cloned

to_table (*a: auto(keyT) const[]*)

to_table returns table<keyT;void>

argument	argument type
a	auto(keyT) const[-1]

will convert an array of key-value tuples into a table<key;value> type. arguments will be cloned

to_table_move (*a: auto(keyT)[]*)

to_table_move returns table<keyT;void>

argument	argument type
a	auto(keyT)[-1]

will convert an array of key-value tuples into a table<key;value> type. arguments will be copied or moved

to_table_move (*a: tuple<auto(keyT);auto(valT)>[]*)

to_table_move returns table<keyT;valT>

argument	argument type
a	tuple<auto(keyT);auto(valT)>[-1]

will convert an array of key-value tuples into a table<key;value> type. arguments will be copied or moved

sort (*a: auto(TT)[] -const|auto(TT)[]# -const*)

sort returns auto

argument	argument type
a	option

sorts an array in ascending order.

sort (*a: array<auto(TT)> -const|array<auto(TT)># -const*)

sort returns auto

argument	argument type
a	option

sorts an array in ascending order.

sort (*a: auto(TT)[] -const|auto(TT)[]# -const; cmp: block<(x:TT const;y:TT const):bool> const*)

sort returns auto

argument	argument type
a	option
cmp	block<(x:TT const;y:TT const):bool> const

sorts an array in ascending order.

sort (*a: array<auto(TT)> -const|array<auto(TT)># -const; cmp: block<(x:TT const;y:TT const):bool> const*)

sort returns auto

argument	argument type
a	option
cmp	block<(x:TT const;y:TT const):bool> const

sorts an array in ascending order.

lock (*a: array<auto(TT)> ==const -const|array<auto(TT)># ==const -const; blk: block<(var x:array<TT># -const):auto> const*)

lock returns auto

argument	argument type
a	option
blk	block<(x:array<TT>#):auto> const

locks array or table for the duration of the block invocation, so that it can't be resized. values can't be pushed or popped, etc.

lock (*a: array<auto(TT)> const ==const|array<auto(TT)> const# ==const const; blk: block<(x:array<TT> const#):auto> const*)

lock returns auto

argument	argument type
a	option const
blk	block<(x:array<TT> const#):auto> const

locks array or table for the duration of the block invocation, so that it can't be resized. values can't be pushed or popped, etc.

find_index (*arr: array<auto(TT)> const|array<auto(TT)> const# const; key: TT const*)

find_index returns auto

argument	argument type
arr	option const
key	TT const

returns index of they key in the array

find_index (*arr: auto(TT) const[ ]|auto(TT) const[ ]# const; key: TT const*)

find_index returns auto

argument	argument type
arr	option const
key	TT const

returns index of they key in the array

find_index (*arr: iterator<auto(TT)> const; key: TT const*)

find_index returns auto

argument	argument type
arr	iterator<auto(TT)> const
key	TT const

returns index of they key in the array

find_index_if (*arr: array<auto(TT)> const|array<auto(TT)> const# const; blk: block<(key:TT const):bool> const*)

find_index_if returns auto

argument	argument type
arr	option const
blk	block<(key:TT const):bool> const

returns index of the key in the array, where key is checked via compare block

find_index_if (*arr: auto(TT) const[ ]|auto(TT) const[ ]# const; blk: block<(key:TT const):bool> const*)

find_index_if returns auto

argument	argument type
arr	option const
blk	block<(key:TT const):bool> const

returns index of the key in the array, where key is checked via compare block

find_index_if (*arr: iterator<auto(TT)> const; blk: block<(key:TT const -&):bool> const*) find_index_if returns auto

argument	argument type
arr	iterator<auto(TT)> const
blk	block<(key:TT const):bool> const

returns index of the key in the array, where key is checked via compare block

has_value (*a: auto const; key: auto const*)

has_value returns auto

argument	argument type
a	auto const
key	auto const

returns true if iterable *a* (array, dim, etc) contains *key*

subarray (*a: auto(TT) const[]; r: range const*)

subarray returns auto

argument	argument type
a	auto(TT) const[-1]
r	range const

returns new array which is copy of a slice of range of the source array

subarray (*a: auto(TT) const[]; r: urange const*)

subarray returns auto

argument	argument type
a	auto(TT) const[-1]
r	urange const

returns new array which is copy of a slice of range of the source array

subarray (*a: array<auto(TT)> const; r: range const*)

subarray returns auto

argument	argument type
a	array<auto(TT)> const
r	range const

returns new array which is copy of a slice of range of the source array

subarray (*a: array<auto(TT)> const; r: urange const*)

subarray returns auto

argument	argument type
a	array<auto(TT)> const
r	urange const

returns new array which is copy of a slice of range of the source array

move_to_ref (*a: auto&; b: auto*)

move_to_ref returns auto

argument	argument type
a	auto&
b	auto

moves *b* into *a*. if *b* is value, it will be copied to *a* instead

clear (*t: table<auto(KT);auto(VT)>*)

clear returns auto

argument	argument type
t	table<auto(KT);auto(VT)>

clear will clear whole table or array *arg*. The size of *arg* after clear is 0.

2.11 das::string manipulation

- *peek (src:$::das_string const implicit;block:block<(arg0:string const#):void> const implicit;context:__context const;line:__lineInfo const) : void*

peek (*src: das_string const implicit; block: block<(arg0:string const#):void> const implicit*)

argument	argument type
src	*builtin::das_string* const implicit
block	block<(string const#):void> const implicit

returns contents of the das::string as temporary string value. this is fastest way to access contents of das::string as string

2.12 String builder

- *write (arg0:$::HashBuilder implicit;arg1:string const implicit) : void*

write (*arg0: HashBuilder implicit; arg1: string const implicit*)

argument	argument type
arg0	*builtin::HashBuilder* implicit
arg1	string const implicit

writes string to a hash-builder.

2.13 Heap reporting

- *heap_bytes_allocated (context:__context const) : uint64*
- *heap_depth (context:__context const) : int*
- *string_heap_bytes_allocated (context:__context const) : uint64*
- *string_heap_depth (context:__context const) : int*
- *string_heap_collect (validate:bool const;context:__context const;at:__lineInfo const) : void*
- *heap_collect (string_heap:bool const;validate:bool const;context:__context const;at:__lineInfo const) : void*
- *string_heap_report (context:__context const;line:__lineInfo const) : void*
- *heap_report (context:__context const;line:__lineInfo const) : void*
- *memory_report (errorsOnly:bool const;context:__context const;lineinfo:__lineInfo const) : void*

heap_bytes_allocated ()

heap_bytes_allocated returns uint64

will return bytes allocated on heap (i.e. really used, not reserved)

heap_depth()

heap_depth returns int

returns number of generations in the regular heap

string_heap_bytes_allocated()

string_heap_bytes_allocated returns uint64

returns number of bytes allocated in the string heap

string_heap_depth()

string_heap_depth returns int

returns number of generations in the string heap

string_heap_collect(*validate: bool const*)

> **Warning:** This is unsafe operation.

argument	argument type
validate	bool const

calls garbage collection on the string heap

heap_collect(*string_heap: bool const; validate: bool const*)

> **Warning:** This is unsafe operation.

argument	argument type
string_heap	bool const
validate	bool const

calls garbage collection on the regular heap

string_heap_report()

reports string heap usage and allocations

heap_report()

reports heap usage and allocations

memory_report(*errorsOnly: bool const*)

argument	argument type
errorsOnly	bool const

reports memory allocation, optionally GC errors only

2.14 GC0 infrastructure

- *gc0_save_ptr (name:string const implicit;data:void? const implicit;context:__context const;line:__lineInfo const) : void*
- *gc0_save_smart_ptr (name:string const implicit;data:smart_ptr<void> const implicit;context:__context const;line:__lineInfo const) : void*
- *gc0_restore_ptr (name:string const implicit;context:__context const) : void?*
- *gc0_restore_smart_ptr (name:string const implicit;context:__context const) : smart_ptr<void>*
- *gc0_reset () : void*

gc0_save_ptr (*name: string const implicit; data: void? const implicit*)

argument	argument type
name	string const implicit
data	void? const implicit

saves pointer to gc0 storage by specifying *name*

gc0_save_smart_ptr (*name: string const implicit; data: smart_ptr<void> const implicit*)

argument	argument type
name	string const implicit
data	smart_ptr<void> const implicit

saves smart_ptr to gc0 storage by specifying *name*

gc0_restore_ptr (*name: string const implicit*)

gc0_restore_ptr returns void?

argument	argument type
name	string const implicit

restores pointer from gc0 storage by *name*

gc0_restore_smart_ptr (*name: string const implicit*)

gc0_restore_smart_ptr returns smart_ptr<void>

argument	argument type
name	string const implicit

restores smart_ptr from gc0 storage *name*

gc0_reset ()

resets gc0 storage. stored pointers will no longer be accessible

2.15 Smart ptr infrastructure

- *move_new (dest:smart_ptr<void>& implicit;src:smart_ptr<void> const implicit;context:__context const;at:__lineInfo const) : void*
- *move (dest:smart_ptr<void>& implicit;src:void? const implicit;context:__context const;at:__lineInfo const) : void*
- *move (dest:smart_ptr<void>& implicit;src:smart_ptr<void>& implicit;context:__context const;at:__lineInfo const) : void*
- *smart_ptr_clone (dest:smart_ptr<void>& implicit;src:void? const implicit;context:__context const;at:__lineInfo const) : void*
- *smart_ptr_clone (dest:smart_ptr<void>& implicit;src:smart_ptr<void> const implicit;context:__context const;at:__lineInfo const) : void*
- *smart_ptr_use_count (ptr:smart_ptr<void> const implicit;context:__context const;at:__lineInfo const) : uint*
- *smart_ptr_is_valid (dest:smart_ptr<void> const implicit) : bool*
- *get_ptr (src:smart_ptr<auto(TT)> const) : TT?*
- *get_const_ptr (src:smart_ptr<auto(TT)> const) : TT? const*
- *add_ptr_ref (src:smart_ptr<auto(TT)> const) : smart_ptr<TT>*

move_new (*dest: smart_ptr<void>& implicit; src: smart_ptr<void> const implicit*)

argument	argument type
dest	smart_ptr<void>& implicit
src	smart_ptr<void> const implicit

Moves the new [[. . .]] value into smart_ptr.

move (*dest: smart_ptr<void>& implicit; src: void? const implicit*)

argument	argument type
dest	smart_ptr<void>& implicit
src	void? const implicit

Moves one smart_ptr into another. Semantic equivalent of move(a,b) => a := null, a <- b

move (*dest: smart_ptr<void>& implicit; src: smart_ptr<void>& implicit*)

argument	argument type
dest	smart_ptr<void>& implicit
src	smart_ptr<void>& implicit

Moves one smart_ptr into another. Semantic equivalent of move(a,b) => a := null, a <- b

smart_ptr_clone (*dest: smart_ptr<void>& implicit; src: void? const implicit*)

argument	argument type
dest	smart_ptr<void>& implicit
src	void? const implicit

clones smart_ptr, internal use-count is incremented

smart_ptr_clone (*dest: smart_ptr<void>& implicit; src: smart_ptr<void> const implicit*)

argument	argument type
dest	smart_ptr<void>& implicit
src	smart_ptr<void> const implicit

clones smart_ptr, internal use-count is incremented

smart_ptr_use_count (*ptr: smart_ptr<void> const implicit*)

smart_ptr_use_count returns uint

argument	argument type
ptr	smart_ptr<void> const implicit

returns internal use-count for the smart_ptr

smart_ptr_is_valid (*dest: smart_ptr<void> const implicit*) smart_ptr_is_valid returns bool

argument	argument type
dest	smart_ptr<void> const implicit

checks if smart pointer points to a valid data.

get_ptr (*src: smart_ptr<auto(TT)> const*)

get_ptr returns TT?

argument	argument type
src	smart_ptr<auto(TT)> const

returns regular pointer from the smart_ptr

get_const_ptr (*src: smart_ptr<auto(TT)> const*)

get_const_ptr returns TT? const

argument	argument type
src	smart_ptr<auto(TT)> const

return constant pointer from regular pointer

add_ptr_ref (*src: smart_ptr<auto(TT)> const*)

add_ptr_ref returns smart_ptr<TT>

argument	argument type
src	smart_ptr<auto(TT)> const

increases reference count of the smart pointer.

2.16 Macro infrastructure

- *is_compiling () : bool*
- *is_compiling_macros () : bool*
- *is_compiling_macros_in_module (name:string const implicit) : bool*
- *is_reporting_compilation_errors () : bool*
- *is_in_completion () : bool*
- *is_folding () : bool*

is_compiling()

is_compiling returns bool

returns true if context is being compiled

is_compiling_macros()

is_compiling_macros returns bool

returns true if context is being compiled and the compiler is currently executing macro pass

is_compiling_macros_in_module(*name: string const implicit*)

is_compiling_macros_in_module returns bool

argument	argument type
name	string const implicit

returns true if context is being compiled, its macro pass, and its in the specific module

is_reporting_compilation_errors()

is_reporting_compilation_errors returns bool

returns true if context failed to compile, and infer pass is reporting compilation errors

is_in_completion()

is_in_completion returns bool

returns true if compiler is currently generating completion, i.e. lexical representation of the program for the text editor's text completion system.

is_folding()

is_folding returns bool

returns true if context is beeing folded, i.e during constant folding pass

2.17 Profiler

- *reset_profiler (context:__context const) : void*
- *dump_profile_info (context:__context const) : void*
- *collect_profile_info (context:__context const) : string*
- *profile (count:int const;category:string const implicit;block:block<> const implicit;context:__context const;line:__lineInfo const) : float*

reset_profiler()

resets counters in the built-in profiler

dump_profile_info()

dumps use counts of all lines collected by built-in profiler

collect_profile_info()

collect_profile_info returns string

enabling collecting of the use counts by built-in profiler

profile (*count: int const; category: string const implicit; block: block<> const implicit*)

profile returns float

argument	argument type
count	int const
category	string const implicit
block	block<> const implicit

profiles specified block by evaluating it *count* times and returns minimal time spent in the block in seconds, as well as prints it.

2.18 System infastructure

- *get_das_root (context:__context const) : string*
- *panic (text:string const implicit;context:__context const;at:__lineInfo const) : void*
- *print (text:string const implicit;context:__context const;at:__lineInfo const) : void*
- *error (text:string const implicit;context:__context const;at:__lineInfo const) : void*
- *sprint (value:any;flags:bitfield<escapeString;namesAndDimensions;typeQualifiers;refAddresses;humanReadable;singleLine> const) : string*
- *sprint_json (value:any;humanReadable:bool const) : string*
- *terminate (context:__context const;at:__lineInfo const) : void*
- *breakpoint () : void*
- *stackwalk (args:bool const;vars:bool const;context:__context const;lineinfo:__lineInfo const) : void*
- *is_in_aot () : bool*
- *to_log (level:int const;text:string const implicit;context:__context const;at:__lineInfo const) : void*
- *to_compiler_log (text:string const implicit;context:__context const;at:__lineInfo const) : void*

get_das_root ()

get_das_root returns string

returns path to where *daslib* and other libraries exist. this is typically root folder of the Daslang main repository

panic (*text: string const implicit*)

argument	argument type
text	string const implicit

will cause panic. The program will be determinated if there is no recover. Panic is not a error handling mechanism and can not be used as such. It is indeed panic, fatal error. It is not supposed that program can completely correctly recover from panic, recover construction is provided so program can try to correcly shut-down or report fatal error. If there is no recover withing script, it will be called in calling eval (in C++ callee code).

print (*text: string const implicit*)

argument	argument type
text	string const implicit

outputs string into current context log output

error (*text: string const implicit*)

argument	argument type
text	string const implicit

similar to 'print' but outputs to context error output

sprint (*value: any; flags: print_flags*)

sprint returns string

argument	argument type
value	any
flags	*print_flags*

similar to 'print' but returns string instead of printing it

sprint_json (*value: any; humanReadable: bool const*)

sprint_json returns string

argument	argument type
value	any
humanReadable	bool const

similar to 'write_json' but skips intermediate representation. this is faster but less flexible

terminate ()

terminates current context execution

breakpoint ()

breakpoint will call os_debugbreakpoint, which is link-time unresolved dependency. It's supposed to call breakpoint in debugger tool, as sample implementation does.

stackwalk (*args: bool const; vars: bool const*)

argument	argument type
args	bool const
vars	bool const

stackwalk prints call stack and local variables values

is_in_aot ()

is_in_aot returns bool

returns true if compiler is currently generating AOT

to_log (*level: int const; text: string const implicit*)

argument	argument type
level	int const
text	string const implicit

similar to print but output goes to the logging infrastructure. *arg0* specifies log level, i.e. **LOG_**… constants

to_compiler_log (*text: string const implicit*)

argument	argument type
text	string const implicit

Output text to compiler log, usually from the macro.

2.19 Memory manipulation

- *variant_index (arg0:variant<> const implicit) : int*
- *set_variant_index (variant:variant<> implicit;index:int const) : void*
- *hash (data:any) : uint64*
- *hash (data:string const implicit) : uint64*
- *memcpy (left:void? const implicit;right:void? const implicit;size:int const) : void*
- *memcmp (left:void? const implicit;right:void? const implicit;size:int const) : int*
- *intptr (p:void? const) : uint64*
- *intptr (p:smart_ptr<auto> const) : uint64*
- *lock_data (a:array<auto(TT)> ==const -const|array<auto(TT)># ==const -const -const;blk:block<(var p:TT?# -const;s:int const):auto> const) : auto*

- *lock_data (a:array<auto(TT)> const ==const|array<auto(TT)> const# ==const const;blk:block<(p:TT const? const#;s:int const):auto> const) : auto*
- *map_to_array (data:void? const;len:int const;blk:block<(var arg:array<auto(TT)># -const):auto> const) : auto*
- *map_to_ro_array (data:void? const;len:int const;blk:block<(arg:array<auto(TT)> const#):auto> const) : auto*

variant_index (*arg0: variant<> const implicit*)

variant_index returns int

argument	argument type
arg0	variant<> const implicit

returns internal index of the variant value

set_variant_index (*variant: variant<> implicit; index: int const*)

Warning: This is unsafe operation.

argument	argument type
variant	variant<> implicit
index	int const

sets internal index of the variant value

hash (*data: any*)

hash returns uint64

argument	argument type
data	any

returns hash value of the *data*. current implementation uses FNV64a hash.

hash (*data: string const implicit*)

hash returns uint64

argument	argument type
data	string const implicit

returns hash value of the *data*. current implementation uses FNV64a hash.

memcpy (*left: void? const implicit; right: void? const implicit; size: int const*)

> **Warning:** This is unsafe operation.

argument	argument type
left	void? const implicit
right	void? const implicit
size	int const

copies *size* bytes of memory from *right* to *left*

memcmp (*left: void? const implicit; right: void? const implicit; size: int const*)

memcmp returns int

> **Warning:** This is unsafe operation.

argument	argument type
left	void? const implicit
right	void? const implicit
size	int const

similar to C 'memcmp', compares *size* bytes of *left* ` and *right* memory. returns -1 if left is less, 1 if left is greater, and 0 if left is same as right

intptr (*p: void? const*)

intptr returns uint64

argument	argument type
p	void? const

returns int64 representation of a pointer

intptr (*p: smart_ptr<auto> const*)

intptr returns uint64

argument	argument type
p	smart_ptr<auto> const

returns int64 representation of a pointer

lock_data (*a: array<auto(TT)> ==const -const|array<auto(TT)># ==const -const; blk: block<(var p:TT?# -const;s:int const):auto> const*)

lock_data returns auto

argument	argument type
a	option
blk	block<(p:TT?#;s:int const):auto> const

locks array and invokes block with a pointer to array's data

lock_data (*a: array<auto(TT)> const ==const|array<auto(TT)> const# ==const const; blk: block<(p:TT const? const#;s:int const):auto> const*)

lock_data returns auto

argument	argument type
a	option const
blk	block<(p:TT const? const#;s:int const):auto> const

locks array and invokes block with a pointer to array's data

map_to_array (*data: void? const; len: int const; blk: block<(var arg:array<auto(TT)># -const):auto> const*)

map_to_array returns auto

> **Warning:** This is unsafe operation.

argument	argument type
data	void? const
len	int const
blk	block<(arg:array<auto(TT)>#):auto> const

builds temporary array from the specified memory

map_to_ro_array (*data: void? const; len: int const; blk: block<(arg:array<auto(TT)> const#):auto> const*)

map_to_ro_array returns auto

Warning: This is unsafe operation.

argument	argument type
data	void? const
len	int const
blk	block<(arg:array<auto(TT)> const#):auto> const

same as *map_to_array* but array is read-only

2.20 Binary serializer

- *binary_save (obj:auto const;subexpr:block<(data:array<uint8> const):void> const) : auto*
- *binary_load (obj:auto -const;data:array<uint8> const) : auto*

binary_save (*obj: auto const; subexpr: block<(data:array<uint8> const):void> const*)

binary_save returns auto

argument	argument type
obj	auto const
subexpr	block<(data:array<uint8> const):void> const

saves any data to array<uint8>. obsolete, use daslib/archive instead

binary_load (*obj: auto; data: array<uint8> const*)

binary_load returns auto

argument	argument type
obj	auto
data	array<uint8> const

loads any data from array<uint8>. obsolete, use daslib/archive instead

2.21 Path and command line

- *get_command_line_arguments () : array<string>*

get_command_line_arguments()

get_command_line_arguments returns array<string>

returns array of command line arguments.

2.22 Time and date

- *get_clock () : $::clock*
- *ref_time_ticks () : int64*
- *get_time_usec (arg0:int64 const) : int*
- *get_time_nsec (arg0:int64 const) : int64*

get_clock()

get_clock returns *builtin::clock*

return a current calendar time. The value returned generally represents the number of seconds since 00:00 hours, Jan 1, 1970 UTC (i.e., the current unix timestamp).

ref_time_ticks()

ref_time_ticks returns int64

returns current time in ticks

get_time_usec(*arg0: int64 const*)

get_time_usec returns int

argument	argument type
arg0	int64 const

returns time interval in usec, since the specified *reft* (usually from *ref_time_ticks*)

get_time_nsec(*arg0: int64 const*)

get_time_nsec returns int64

argument	argument type
arg0	int64 const

returns time interval in nsec, since the specified *reft* (usually from *ref_time_ticks*)

2.23 Lock checking

- *lock_count (array:array const implicit) : int*
- *set_verify_array_locks (array:array implicit;check:bool const) : bool*
- *set_verify_table_locks (table:table implicit;check:bool const) : bool*

lock_count (*array: array const implicit*)

lock_count returns int

argument	argument type
array	array const implicit

returns internal lock count for the array or table

set_verify_array_locks (*array: array implicit; check: bool const*)

set_verify_array_locks returns bool

Warning: This is unsafe operation.

argument	argument type
array	array implicit
check	bool const

runtime optimization, which indicates that the array does not need lock checks.

set_verify_table_locks (*table: table implicit; check: bool const*)

set_verify_table_locks returns bool

Warning: This is unsafe operation.

argument	argument type
table	table implicit
check	bool const

runtime optimization, which indicates that the table does not need lock checks.

2.24 Lock checking internals

- *_move_with_lockcheck (a:auto(valA)& -const;b:auto(valB)& -const) : auto*
- *_return_with_lockcheck (a:auto(valT)& ==const -const) : valT&*
- *_return_with_lockcheck (a:auto(valT) const& ==const) : valT&*
- *_at_with_lockcheck (Tab:table<auto(keyT);auto(valT)> -const;at:keyT const|keyT const# const) : valT&*

_move_with_lockcheck (*a: auto(valA)&; b: auto(valB)&*)

_move_with_lockcheck returns auto

argument	argument type
a	auto(valA)&
b	auto(valB)&

moves *b* into *a*, checks if *a* or *b* is locked, recursively for each lockable element of a and b

_return_with_lockcheck (*a: auto(valT)& ==const*)

_return_with_lockcheck returns valT&

argument	argument type
a	auto(valT)&!

returns *a*. check if *a* is locked, recursively for each lockable element of a

_return_with_lockcheck (*a: auto(valT) const& ==const*)

_return_with_lockcheck returns valT&

argument	argument type
a	auto(valT) const&!

returns *a*. check if *a* is locked, recursively for each lockable element of a

_at_with_lockcheck (*Tab: table<auto(keyT);auto(valT)>; at: keyT const|keyT const# const*)

_at_with_lockcheck returns valT&

argument	argument type
Tab	table<auto(keyT);auto(valT)>
at	option const

returns element of the table *Tab*, also checks if *Tab* is locked, recursively for each lockable element of *Tab*

2.25 Bit operations

- *clz (bits:uint const) : uint*
- *clz (bits:uint64 const) : uint64*
- *ctz (bits:uint const) : uint*
- *ctz (bits:uint64 const) : uint64*
- *popcnt (bits:uint const) : uint*
- *popcnt (bits:uint64 const) : uint64*

clz (*bits: uint const*)

clz returns uint

argument	argument type
bits	uint const

count leading zeros

clz (*bits: uint64 const*)

clz returns uint64

argument	argument type
bits	uint64 const

count leading zeros

ctz (*bits: uint const*)

ctz returns uint

argument	argument type
bits	uint const

count trailing zeros

ctz (*bits: uint64 const*)

ctz returns uint64

argument	argument type
bits	uint64 const

count trailing zeros

popcnt (*bits: uint const*)

popcnt returns uint

argument	argument type
bits	uint const

count number of set bits

popcnt (*bits: uint64 const*)

popcnt returns uint64

argument	argument type
bits	uint64 const

count number of set bits

2.26 Intervals

- *interval (arg0:int const;arg1:int const) : range*
- *interval (arg0:uint const;arg1:uint const) : urange*
- *interval (arg0:int64 const;arg1:int64 const) : range64*
- *interval (arg0:uint64 const;arg1:uint64 const) : urange64*

interval (*arg0: int const; arg1: int const*)

interval returns range

argument	argument type
arg0	int const
arg1	int const

returns range(‘arg0’,’arg1’)

interval (*arg0: uint const; arg1: uint const*)

interval returns urange

argument	argument type
arg0	uint const
arg1	uint const

returns range('arg0','arg1')

interval (*arg0: int64 const; arg1: int64 const*)

interval returns range64

argument	argument type
arg0	int64 const
arg1	int64 const

returns range('arg0','arg1')

interval (*arg0: uint64 const; arg1: uint64 const*)

interval returns urange64

argument	argument type
arg0	uint64 const
arg1	uint64 const

returns range('arg0','arg1')

2.27 RTTI

- *class_rtti_size (ptr:void? const implicit) : int*

class_rtti_size (*ptr: void? const implicit*)

class_rtti_size returns int

argument	argument type
ptr	void? const implicit

returns size of specific TypeInfo for the class

2.28 Lock verification

- *set_verify_context_locks (check:bool const;context:__context const) : bool*

set_verify_context_locks (*check: bool const*)

set_verify_context_locks returns bool

Warning: This is unsafe operation.

argument	argument type
check	bool const

Enables or disables array or table lock runtime verification per context

2.29 Initialization and finalization

- *using (arg0:block<(var arg0:$::das_string explicit):void> const implicit) : void*

using (*arg0: block<(var arg0:das_string explicit):void> const implicit*)

argument	argument type
arg0	block<(*builtin::das_string*):void> const implicit

Creates temporary das_string.

2.30 Algorithms

- *count (start:int const;step:int const;context:__context const) : iterator<int>*
- *ucount (start:uint const;step:uint const;context:__context const) : iterator<uint>*
- *iter_range (foo:auto const) : auto*
- *swap (a:auto(TT)& -const;b:auto(TT)& -const) : auto*

count (*start: int const; step: int const*)

count returns iterator<int>

argument	argument type
start	int const
step	int const

returns iterator which iterates from *start* value by incrementing it by *step* value. It is the intended way to have counter together with other values in the *for* loop.

ucount (*start: uint const; step: uint const*)

ucount returns iterator<uint>

argument	argument type
start	uint const
step	uint const

returns iterator which iterates from *start* value by incrementing it by *step* value. It is the intended way to have counter together with other values in the *for* loop.

iter_range (*foo: auto const*)

iter_range returns auto

argument	argument type
foo	auto const

returns range(*foo*)

swap (*a: auto(TT)&; b: auto(TT)&*)

swap returns auto

argument	argument type
a	auto(TT)&
b	auto(TT)&

swaps two values *a* and 'b'

2.31 Memset

- *memset8 (left:void? const implicit;value:uint8 const;count:int const) : void*
- *memset16 (left:void? const implicit;value:uint16 const;count:int const) : void*
- *memset32 (left:void? const implicit;value:uint const;count:int const) : void*
- *memset64 (left:void? const implicit;value:uint64 const;count:int const) : void*
- *memset128 (left:void? const implicit;value:uint4 const;count:int const) : void*

memset8 (*left: void? const implicit; value: uint8 const; count: int const*)

> **Warning:** This is unsafe operation.

argument	argument type
left	void? const implicit
value	uint8 const
count	int const

Effecitvely C memset.

memset16 (*left: void? const implicit; value: uint16 const; count: int const*)

> **Warning:** This is unsafe operation.

argument	argument type
left	void? const implicit
value	uint16 const
count	int const

Similar to memset, but fills values with 16 bit words.

memset32 (*left: void? const implicit; value: uint const; count: int const*)

> **Warning:** This is unsafe operation.

argument	argument type
left	void? const implicit
value	uint const
count	int const

Similar to memset, but fills values with 32 bit words.

memset64 (*left: void? const implicit; value: uint64 const; count: int const*)

> **Warning:** This is unsafe operation.

argument	argument type
left	void? const implicit
value	uint64 const
count	int const

Similar to memset, but fills values with 64 bit words.

memset128 (*left: void? const implicit; value: uint4 const; count: int const*)

> **Warning:** This is unsafe operation.

argument	argument type
left	void? const implicit
value	uint4 const
count	int const

Similar to memset, but fills values with 128 bit vector type values.

2.32 Malloc

- *malloc (size:uint64 const) : void?*
- *free (ptr:void? const implicit) : void*
- *malloc_usable_size (ptr:void? const implicit) : uint64*

malloc (*size: uint64 const*)

malloc returns void?

> **Warning:** This is unsafe operation.

argument	argument type
size	uint64 const

C-style malloc

free (*ptr: void? const implicit*)

Warning: This is unsafe operation.

argument	argument type
ptr	void? const implicit

C-style free to be coupled with C-style malloc

malloc_usable_size (*ptr: void? const implicit*)

malloc_usable_size returns uint64

Warning: This is unsafe operation.

argument	argument type
ptr	void? const implicit

returns size of the allocated memory block

2.33 Uncategorized

is_intern_strings ()

is_intern_strings returns bool

returns true if string interning is enabled

build_hash (*block: block<(var arg0:HashBuilder):void> const implicit*)

build_hash returns uint64

argument	argument type
block	block<(*builtin::HashBuilder*):void> const implicit

returns hash value out of hash-builder.

eval_main_loop (*block: block<bool> const implicit*)

argument	argument type
block	block<> const implicit

executes main loop for the application. has specific implementation in EMSCRIPTEN, otherwise invoke until false.

`remove_value` (*arr: array<auto(TT)> -const|array<auto(TT)># -const; key: TT const*)

remove_value returns bool

argument	argument type
arr	option
key	TT const

removes first occurance of the key from the array.

CHAPTER

THREE

MATH LIBRARY

Floating point math in general is not bit-precise. Compiler can optimize permutations, replace divisions with multiplications, and some of functions are not bit-exact. If you need precise math use double precision type. All functions and symbols are in "math" module, use require to get access to it.

```
require math
```

3.1 Constants

`PI = 3.14159f`

The ratio of a circle's circumference to its diameter. π

`FLT_EPSILON = 1.19209e-07f`

the difference between 1 and the smallest floating point number of type float that is greater than 1.

`DBL_EPSILON = 2.22045e-16lf`

the difference between 1 and the smallest double precision floating point number of type double that is greater than 1.

3.2 Handled structures

float4x4

float4x4 fields are

z	float4
w	float4
y	float4
x	float4

floating point matrix with 4 rows and 4 columns

float3x4

float3x4 fields are

z	float3
w	float3
y	float3
x	float3

floating point matrix with 4 rows and 3 columns

float3x3

float3x3 fields are

z	float3
y	float3
x	float3

floating point matrix with 3 rows and 3 columns

3.3 all numerics (uint*, int*, float*, double)

- *min (x:int const;y:int const) : int*
- *max (x:int const;y:int const) : int*
- *min (x:int2 const;y:int2 const) : int2*
- *max (x:int2 const;y:int2 const) : int2*
- *min (x:int3 const;y:int3 const) : int3*
- *max (x:int3 const;y:int3 const) : int3*
- *min (x:int4 const;y:int4 const) : int4*
- *max (x:int4 const;y:int4 const) : int4*
- *min (x:uint const;y:uint const) : uint*
- *max (x:uint const;y:uint const) : uint*
- *min (x:uint2 const;y:uint2 const) : uint2*
- *max (x:uint2 const;y:uint2 const) : uint2*
- *min (x:uint3 const;y:uint3 const) : uint3*
- *max (x:uint3 const;y:uint3 const) : uint3*
- *min (x:uint4 const;y:uint4 const) : uint4*
- *max (x:uint4 const;y:uint4 const) : uint4*

- *min (x:float const;y:float const) : float*
- *max (x:float const;y:float const) : float*
- *min (x:float2 const;y:float2 const) : float2*
- *max (x:float2 const;y:float2 const) : float2*
- *min (x:float3 const;y:float3 const) : float3*
- *max (x:float3 const;y:float3 const) : float3*
- *min (x:float4 const;y:float4 const) : float4*
- *max (x:float4 const;y:float4 const) : float4*
- *min (x:double const;y:double const) : double*
- *max (x:double const;y:double const) : double*
- *min (x:int64 const;y:int64 const) : int64*
- *max (x:int64 const;y:int64 const) : int64*
- *min (x:uint64 const;y:uint64 const) : uint64*
- *max (x:uint64 const;y:uint64 const) : uint64*

min (*x: int const; y: int const*)

min returns int

argument	argument type
x	int const
y	int const

returns the minimum of x and y

max (*x: int const; y: int const*)

max returns int

argument	argument type
x	int const
y	int const

returns the maximum of x and y

min (*x: int2 const; y: int2 const*)

min returns int2

argument	argument type
x	int2 const
y	int2 const

returns the minimum of x and y

max (*x: int2 const; y: int2 const*)

max returns int2

argument	argument type
x	int2 const
y	int2 const

returns the maximum of x and y

min (*x: int3 const; y: int3 const*)

min returns int3

argument	argument type
x	int3 const
y	int3 const

returns the minimum of x and y

max (*x: int3 const; y: int3 const*)

max returns int3

argument	argument type
x	int3 const
y	int3 const

returns the maximum of x and y

min (*x: int4 const; y: int4 const*)

min returns int4

argument	argument type
x	int4 const
y	int4 const

returns the minimum of x and y

max (*x: int4 const; y: int4 const*)

max returns int4

argument	argument type
x	int4 const
y	int4 const

returns the maximum of x and y

min (*x: uint const; y: uint const*)

min returns uint

argument	argument type
x	uint const
y	uint const

returns the minimum of x and y

max (*x: uint const; y: uint const*)

max returns uint

argument	argument type
x	uint const
y	uint const

returns the maximum of x and y

min (*x: uint2 const; y: uint2 const*)

min returns uint2

argument	argument type
x	uint2 const
y	uint2 const

returns the minimum of x and y

max (*x: uint2 const; y: uint2 const*)

max returns uint2

argument	argument type
x	uint2 const
y	uint2 const

returns the maximum of x and y

min (*x: uint3 const; y: uint3 const*)

min returns uint3

argument	argument type
x	uint3 const
y	uint3 const

returns the minimum of x and y

max (*x: uint3 const; y: uint3 const*)

max returns uint3

argument	argument type
x	uint3 const
y	uint3 const

returns the maximum of x and y

min (*x: uint4 const; y: uint4 const*)

min returns uint4

argument	argument type
x	uint4 const
y	uint4 const

returns the minimum of x and y

max (*x: uint4 const; y: uint4 const*)

max returns uint4

argument	argument type
x	uint4 const
y	uint4 const

returns the maximum of x and y

min (*x: float const; y: float const*)

min returns float

argument	argument type
x	float const
y	float const

returns the minimum of x and y

max (*x: float const; y: float const*)

max returns float

argument	argument type
x	float const
y	float const

returns the maximum of x and y

min (*x: float2 const; y: float2 const*)

min returns float2

argument	argument type
x	float2 const
y	float2 const

returns the minimum of x and y

max (*x: float2 const; y: float2 const*)

max returns float2

argument	argument type
x	float2 const
y	float2 const

returns the maximum of x and y

min (*x: float3 const; y: float3 const*)

min returns float3

argument	argument type
x	float3 const
y	float3 const

returns the minimum of x and y

max (*x: float3 const; y: float3 const*)

max returns float3

argument	argument type
x	float3 const
y	float3 const

returns the maximum of x and y

min (*x: float4 const; y: float4 const*)

min returns float4

argument	argument type
x	float4 const
y	float4 const

returns the minimum of x and y

max (*x: float4 const; y: float4 const*)

max returns float4

argument	argument type
x	float4 const
y	float4 const

returns the maximum of x and y

min (*x: double const; y: double const*)

min returns double

argument	argument type
x	double const
y	double const

returns the minimum of x and y

max (*x: double const; y: double const*)

max returns double

argument	argument type
x	double const
y	double const

returns the maximum of x and y

min (*x: int64 const; y: int64 const*)

min returns int64

argument	argument type
x	int64 const
y	int64 const

returns the minimum of x and y

max (*x: int64 const; y: int64 const*)

max returns int64

argument	argument type
x	int64 const
y	int64 const

returns the maximum of x and y

min (*x: uint64 const; y: uint64 const*)

min returns uint64

argument	argument type
x	uint64 const
y	uint64 const

returns the minimum of x and y

max (*x: uint64 const; y: uint64 const*)

max returns uint64

argument	argument type
x	uint64 const
y	uint64 const

returns the maximum of x and y

3.4 float* and double

- *sin (x:float const) : float*
- *cos (x:float const) : float*
- *tan (x:float const) : float*
- *atan (x:float const) : float*
- *asin (x:float const) : float*
- *acos (x:float const) : float*
- *atan2 (y:float const;x:float const) : float*
- *sin (x:float2 const) : float2*
- *cos (x:float2 const) : float2*
- *tan (x:float2 const) : float2*
- *atan (x:float2 const) : float2*
- *asin (x:float2 const) : float2*
- *acos (x:float2 const) : float2*
- *atan2 (y:float2 const;x:float2 const) : float2*
- *sin (x:float3 const) : float3*
- *cos (x:float3 const) : float3*
- *tan (x:float3 const) : float3*
- *atan (x:float3 const) : float3*
- *asin (x:float3 const) : float3*
- *acos (x:float3 const) : float3*
- *atan2 (y:float3 const;x:float3 const) : float3*
- *sin (x:float4 const) : float4*
- *cos (x:float4 const) : float4*
- *tan (x:float4 const) : float4*
- *atan (x:float4 const) : float4*
- *asin (x:float4 const) : float4*
- *acos (x:float4 const) : float4*
- *atan2 (y:float4 const;x:float4 const) : float4*
- *exp (x:float const) : float*
- *log (x:float const) : float*
- *exp2 (x:float const) : float*
- *log2 (x:float const) : float*
- *rcp (x:float const) : float*
- *pow (x:float const;y:float const) : float*

- *exp (x:float2 const) : float2*
- *log (x:float2 const) : float2*
- *exp2 (x:float2 const) : float2*
- *log2 (x:float2 const) : float2*
- *rcp (x:float2 const) : float2*
- *pow (x:float2 const;y:float2 const) : float2*
- *exp (x:float3 const) : float3*
- *log (x:float3 const) : float3*
- *exp2 (x:float3 const) : float3*
- *log2 (x:float3 const) : float3*
- *rcp (x:float3 const) : float3*
- *pow (x:float3 const;y:float3 const) : float3*
- *exp (x:float4 const) : float4*
- *log (x:float4 const) : float4*
- *exp2 (x:float4 const) : float4*
- *log2 (x:float4 const) : float4*
- *rcp (x:float4 const) : float4*
- *pow (x:float4 const;y:float4 const) : float4*
- *floor (x:float const) : float*
- *ceil (x:float const) : float*
- *sqrt (x:float const) : float*
- *saturate (x:float const) : float*
- *floor (x:float2 const) : float2*
- *ceil (x:float2 const) : float2*
- *sqrt (x:float2 const) : float2*
- *saturate (x:float2 const) : float2*
- *floor (x:float3 const) : float3*
- *ceil (x:float3 const) : float3*
- *sqrt (x:float3 const) : float3*
- *saturate (x:float3 const) : float3*
- *floor (x:float4 const) : float4*
- *ceil (x:float4 const) : float4*
- *sqrt (x:float4 const) : float4*
- *saturate (x:float4 const) : float4*
- *abs (x:int const) : int*
- *sign (x:int const) : int*

- *abs (x:int2 const) : int2*
- *sign (x:int2 const) : int2*
- *abs (x:int3 const) : int3*
- *sign (x:int3 const) : int3*
- *abs (x:int4 const) : int4*
- *sign (x:int4 const) : int4*
- *abs (x:uint const) : uint*
- *sign (x:uint const) : uint*
- *abs (x:uint2 const) : uint2*
- *sign (x:uint2 const) : uint2*
- *abs (x:uint3 const) : uint3*
- *sign (x:uint3 const) : uint3*
- *abs (x:uint4 const) : uint4*
- *sign (x:uint4 const) : uint4*
- *abs (x:float const) : float*
- *sign (x:float const) : float*
- *abs (x:float2 const) : float2*
- *sign (x:float2 const) : float2*
- *abs (x:float3 const) : float3*
- *sign (x:float3 const) : float3*
- *abs (x:float4 const) : float4*
- *sign (x:float4 const) : float4*
- *abs (x:double const) : double*
- *sign (x:double const) : double*
- *abs (x:int64 const) : int64*
- *sign (x:int64 const) : int64*
- *abs (x:uint64 const) : uint64*
- *sign (x:uint64 const) : uint64*
- *is_nan (x:float const) : bool*
- *is_finite (x:float const) : bool*
- *is_nan (x:double const) : bool*
- *is_finite (x:double const) : bool*
- *sqrt (x:double const) : double*
- *exp (x:double const) : double*
- *rcp (x:double const) : double*
- *log (x:double const) : double*

- *pow (x:double const;y:double const) : double*
- *exp2 (x:double const) : double*
- *log2 (x:double const) : double*
- *sin (x:double const) : double*
- *cos (x:double const) : double*
- *asin (x:double const) : double*
- *acos (x:double const) : double*
- *tan (x:double const) : double*
- *atan (x:double const) : double*
- *atan2 (y:double const;x:double const) : double*
- *sincos (x:float const;s:float& implicit;c:float& implicit) : void*
- *sincos (x:double const;s:double& implicit;c:double& implicit) : void*

sin (*x: float const*)

sin returns float

argument	argument type
x	float const

returns the sine of x

cos (*x: float const*)

cos returns float

argument	argument type
x	float const

returns the cosine of x

tan (*x: float const*)

tan returns float

argument	argument type
x	float const

returns the tangent of x

atan (*x: float const*)

atan returns float

argument	argument type
x	float const

returns the arctangent of x

asin (*x: float const*)

asin returns float

argument	argument type
x	float const

returns the arcsine of x

acos (*x: float const*)

acos returns float

argument	argument type
x	float const

returns the arccosine of x

atan2 (*y: float const; x: float const*)

atan2 returns float

argument	argument type
y	float const
x	float const

returns the arctangent of y/x

sin (*x: float2 const*)

sin returns float2

argument	argument type
x	float2 const

returns the sine of x

cos (*x: float2 const*)

cos returns float2

argument	argument type
x	float2 const

returns the cosine of x

tan (*x: float2 const*)

tan returns float2

argument	argument type
x	float2 const

returns the tangent of x

atan (*x: float2 const*)

atan returns float2

argument	argument type
x	float2 const

returns the arctangent of x

asin (*x: float2 const*)

asin returns float2

argument	argument type
x	float2 const

returns the arcsine of x

acos (*x: float2 const*)

acos returns float2

argument	argument type
x	float2 const

returns the arccosine of x

atan2 (*y: float2 const; x: float2 const*) atan2 returns float2

argument	argument type
y	float2 const
x	float2 const

returns the arctangent of y/x

sin (*x: float3 const*)

sin returns float3

argument	argument type
x	float3 const

returns the sine of x

cos (*x: float3 const*)

cos returns float3

argument	argument type
x	float3 const

returns the cosine of x

tan (*x: float3 const*)

tan returns float3

argument	argument type
x	float3 const

returns the tangent of x

atan (*x: float3 const*)

atan returns float3

argument	argument type
x	float3 const

returns the arctangent of x

asin (*x: float3 const*)

asin returns float3

argument	argument type
x	float3 const

returns the arcsine of x

acos (*x: float3 const*)

acos returns float3

argument	argument type
x	float3 const

returns the arccosine of x

atan2 (*y: float3 const; x: float3 const*)

atan2 returns float3

argument	argument type
y	float3 const
x	float3 const

returns the arctangent of y/x

sin (*x: float4 const*)

sin returns float4

argument	argument type
x	float4 const

returns the sine of x

cos (*x: float4 const*)

cos returns float4

argument	argument type
x	float4 const

returns the cosine of x

tan (*x: float4 const*)

tan returns float4

argument	argument type
x	float4 const

returns the tangent of x

atan (*x: float4 const*)

atan returns float4

argument	argument type
x	float4 const

returns the arctangent of x

asin (*x: float4 const*)

asin returns float4

argument	argument type
x	float4 const

returns the arcsine of x

acos (*x: float4 const*)

acos returns float4

argument	argument type
x	float4 const

returns the arccosine of x

atan2 (*y: float4 const; x: float4 const*)

atan2 returns float4

argument	argument type
y	float4 const
x	float4 const

returns the arctangent of y/x

exp (*x: float const*)

exp returns float

argument	argument type
x	float const

returns the e^x value of x

log (*x: float const*)

log returns float

argument	argument type
x	float const

returns the natural logarithm of x

exp2 (*x: float const*)

exp2 returns float

argument	argument type
x	float const

returns the 2^x value of x

log2 (*x: float const*)

log2 returns float

argument	argument type
x	float const

returns the logarithm base-2 of x

rcp (*x: float const*)

rcp returns float

argument	argument type
x	float const

returns the 1/x

pow (*x: float const; y: float const*) pow returns float

argument	argument type
x	float const
y	float const

returns x raised to the power of y

exp (*x: float2 const*)

exp returns float2

argument	argument type
x	float2 const

returns the e^x value of x

log (*x: float2 const*)

log returns float2

argument	argument type
x	float2 const

returns the natural logarithm of x

exp2 (*x: float2 const*)

exp2 returns float2

argument	argument type
x	float2 const

returns the 2^x value of x

log2 (*x: float2 const*)

log2 returns float2

argument	argument type
x	float2 const

returns the logarithm base-2 of x

rcp (*x: float2 const*)

rcp returns float2

argument	argument type
x	float2 const

returns the 1/x

pow (*x: float2 const; y: float2 const*)

pow returns float2

argument	argument type
x	float2 const
y	float2 const

returns x raised to the power of y

exp (*x: float3 const*)

exp returns float3

argument	argument type
x	float3 const

returns the e^x value of x

log (*x: float3 const*)

log returns float3

argument	argument type
x	float3 const

returns the natural logarithm of x

exp2 (*x: float3 const*)

exp2 returns float3

argument	argument type
x	float3 const

returns the 2^x value of x

log2 (*x: float3 const*)

log2 returns float3

argument	argument type
x	float3 const

returns the logarithm base-2 of x

rcp (*x: float3 const*)

rcp returns float3

argument	argument type
x	float3 const

returns the 1/x

pow (*x: float3 const; y: float3 const*)

pow returns float3

argument	argument type
x	float3 const
y	float3 const

returns x raised to the power of y

exp (*x: float4 const*)

exp returns float4

argument	argument type
x	float4 const

returns the e^x value of x

log (*x: float4 const*)

log returns float4

argument	argument type
x	float4 const

returns the natural logarithm of x

exp2 (*x: float4 const*)

exp2 returns float4

argument	argument type
x	float4 const

returns the 2^x value of x

log2 (*x: float4 const*)

log2 returns float4

argument	argument type
x	float4 const

returns the logarithm base-2 of x

rcp (*x: float4 const*)

rcp returns float4

argument	argument type
x	float4 const

returns the 1/x

pow (*x: float4 const; y: float4 const*)

pow returns float4

argument	argument type
x	float4 const
y	float4 const

returns x raised to the power of y

floor (*x: float const*)

floor returns float

argument	argument type
x	float const

returns a float value representing the largest integer that is less than or equal to x

ceil (*x: float const*)

ceil returns float

argument	argument type
x	float const

returns a float value representing the smallest integer (type is still float) that is greater than or equal to arg0

sqrt (*x: float const*)

sqrt returns float

argument	argument type
x	float const

returns the square root of x

saturate (*x: float const*)

saturate returns float

argument	argument type
x	float const

returns a clamped to [0..1] inclusive range x

floor (*x: float2 const*)

floor returns float2

argument	argument type
x	float2 const

returns a float value representing the largest integer that is less than or equal to x

ceil (*x: float2 const*)

ceil returns float2

argument	argument type
x	float2 const

returns a float value representing the smallest integer (type is still float) that is greater than or equal to arg0

sqrt (*x: float2 const*)

sqrt returns float2

argument	argument type
x	float2 const

returns the square root of x

saturate (*x: float2 const*)

saturate returns float2

argument	argument type
x	float2 const

returns a clamped to [0..1] inclusive range x

floor (*x: float3 const*)

floor returns float3

argument	argument type
x	float3 const

returns a float value representing the largest integer that is less than or equal to x

ceil (*x: float3 const*)

ceil returns float3

argument	argument type
x	float3 const

returns a float value representing the smallest integer (type is still float) that is greater than or equal to arg0

sqrt (*x: float3 const*)

sqrt returns float3

argument	argument type
x	float3 const

returns the square root of x

saturate (*x: float3 const*)

saturate returns float3

argument	argument type
x	float3 const

returns a clamped to [0..1] inclusive range x

floor (*x: float4 const*)

floor returns float4

argument	argument type
x	float4 const

returns a float value representing the largest integer that is less than or equal to x

ceil (*x: float4 const*)

ceil returns float4

argument	argument type
x	float4 const

returns a float value representing the smallest integer (type is still float) that is greater than or equal to arg0

sqrt (*x: float4 const*)

sqrt returns float4

argument	argument type
x	float4 const

returns the square root of x

saturate (*x: float4 const*)

saturate returns float4

argument	argument type
x	float4 const

returns a clamped to [0..1] inclusive range x

abs (*x: int const*)

abs returns int

argument	argument type
x	int const

returns the absolute value of x

sign (*x: int const*)

sign returns int

argument	argument type
x	int const

returns sign of x, or 0 if x == 0

abs (*x: int2 const*)

abs returns int2

argument	argument type
x	int2 const

returns the absolute value of x

sign (*x: int2 const*)

sign returns int2

argument	argument type
x	int2 const

returns sign of x, or 0 if x == 0

abs (*x: int3 const*)

abs returns int3

argument	argument type
x	int3 const

returns the absolute value of x

sign (*x: int3 const*)

sign returns int3

argument	argument type
x	int3 const

returns sign of x, or 0 if x == 0

abs (*x: int4 const*)

abs returns int4

argument	argument type
x	int4 const

returns the absolute value of x

sign (*x: int4 const*)

sign returns int4

argument	argument type
x	int4 const

returns sign of x, or 0 if x == 0

abs (*x: uint const*)

abs returns uint

argument	argument type
x	uint const

returns the absolute value of x

sign (*x: uint const*)

sign returns uint

argument	argument type
x	uint const

returns sign of x, or 0 if x == 0

abs (*x: uint2 const*)

abs returns uint2

argument	argument type
x	uint2 const

returns the absolute value of x

sign (*x: uint2 const*)

sign returns uint2

argument	argument type
x	uint2 const

returns sign of x, or 0 if x == 0

abs (*x: uint3 const*)

abs returns uint3

argument	argument type
x	uint3 const

returns the absolute value of x

sign (*x: uint3 const*)

sign returns uint3

argument	argument type
x	uint3 const

returns sign of x, or 0 if x == 0

abs (*x: uint4 const*)

abs returns uint4

argument	argument type
x	uint4 const

returns the absolute value of x

sign (*x: uint4 const*)

sign returns uint4

argument	argument type
x	uint4 const

returns sign of x, or 0 if x == 0

abs (*x: float const*)

abs returns float

argument	argument type
x	float const

returns the absolute value of x

sign (*x: float const*)

sign returns float

argument	argument type
x	float const

returns sign of x, or 0 if x == 0

abs (*x: float2 const*)

abs returns float2

argument	argument type
x	float2 const

returns the absolute value of x

sign (*x: float2 const*)

sign returns float2

argument	argument type
x	float2 const

returns sign of x, or 0 if x == 0

abs (*x: float3 const*)

abs returns float3

argument	argument type
x	float3 const

returns the absolute value of x

sign (*x: float3 const*)

sign returns float3

argument	argument type
x	float3 const

returns sign of x, or 0 if x == 0

abs (*x: float4 const*)

abs returns float4

argument	argument type
x	float4 const

returns the absolute value of x

sign (*x: float4 const*)

sign returns float4

argument	argument type
x	float4 const

returns sign of x, or 0 if x == 0

abs (*x: double const*)

abs returns double

argument	argument type
x	double const

returns the absolute value of x

sign (*x: double const*)

sign returns double

argument	argument type
x	double const

returns sign of x, or 0 if x == 0

abs (*x: int64 const*)

abs returns int64

argument	argument type
x	int64 const

returns the absolute value of x

sign (*x: int64 const*)

sign returns int64

argument	argument type
x	int64 const

returns sign of x, or 0 if x == 0

abs (*x: uint64 const*)

abs returns uint64

argument	argument type
x	uint64 const

returns the absolute value of x

sign (*x: uint64 const*)

sign returns uint64

argument	argument type
x	uint64 const

returns sign of x, or 0 if x == 0

is_nan (*x: float const*)

is_nan returns bool

argument	argument type
x	float const

Returns true if *x* is NaN (not a number)

is_finite (*x: float const*)

is_finite returns bool

argument	argument type
x	float const

Returns true if *x* is not a negative or positive infinity

is_nan (*x: double const*)

is_nan returns bool

argument	argument type
x	double const

Returns true if *x* is NaN (not a number)

is_finite (*x: double const*)

is_finite returns bool

argument	argument type
x	double const

Returns true if *x* is not a negative or positive infinity

sqrt (*x: double const*)

sqrt returns double

argument	argument type
x	double const

returns the square root of x

exp (*x: double const*)

exp returns double

argument	argument type
x	double const

returns the e^x value of x

rcp (*x: double const*)

rcp returns double

argument	argument type
x	double const

returns the 1/x

log (*x: double const*)

log returns double

argument	argument type
x	double const

returns the natural logarithm of x

pow (*x: double const; y: double const*)

pow returns double

argument	argument type
x	double const
y	double const

returns x raised to the power of y

exp2 (*x: double const*)

exp2 returns double

argument	argument type
x	double const

returns the 2^x value of x

log2 (*x: double const*)

log2 returns double

argument	argument type
x	double const

returns the logarithm base-2 of x

sin (*x: double const*)

sin returns double

argument	argument type
x	double const

returns the sine of x

cos (*x: double const*)

cos returns double

argument	argument type
x	double const

returns the cosine of x

asin (*x: double const*)

asin returns double

argument	argument type
x	double const

returns the arcsine of x

acos (*x: double const*)

acos returns double

argument	argument type
x	double const

returns the arccosine of x

tan (*x: double const*)

tan returns double

argument	argument type
x	double const

returns the tangent of x

atan (*x: double const*)

atan returns double

argument	argument type
x	double const

returns the arctangent of x

atan2 (*y: double const; x: double const*)

atan2 returns double

argument	argument type
y	double const
x	double const

returns the arctangent of y/x

sincos (*x: float const; s: float& implicit; c: float& implicit*)

argument	argument type
x	float const
s	float& implicit
c	float& implicit

returns oth sine and cosine of x

sincos (*x: double const; s: double& implicit; c: double& implicit*)

argument	argument type
x	double const
s	double& implicit
c	double& implicit

returns oth sine and cosine of x

3.5 float* only

- *atan_est (x:float const) : float*
- *atan2_est (y:float const;x:float const) : float*
- *atan_est (x:float2 const) : float2*
- *atan2_est (y:float2 const;x:float2 const) : float2*
- *atan_est (x:float3 const) : float3*
- *atan2_est (y:float3 const;x:float3 const) : float3*
- *atan_est (x:float4 const) : float4*
- *atan2_est (y:float4 const;x:float4 const) : float4*
- *rcp_est (x:float const) : float*
- *rcp_est (x:float2 const) : float2*
- *rcp_est (x:float3 const) : float3*
- *rcp_est (x:float4 const) : float4*
- *fract (x:float const) : float*
- *rsqrt (x:float const) : float*
- *rsqrt_est (x:float const) : float*
- *fract (x:float2 const) : float2*
- *rsqrt (x:float2 const) : float2*
- *rsqrt_est (x:float2 const) : float2*
- *fract (x:float3 const) : float3*
- *rsqrt (x:float3 const) : float3*
- *rsqrt_est (x:float3 const) : float3*
- *fract (x:float4 const) : float4*
- *rsqrt (x:float4 const) : float4*
- *rsqrt_est (x:float4 const) : float4*
- *floori (x:float const) : int*
- *ceili (x:float const) : int*

- *roundi (x:float const) : int*
- *trunci (x:float const) : int*
- *floori (x:double const) : int*
- *ceili (x:double const) : int*
- *roundi (x:double const) : int*
- *trunci (x:double const) : int*
- *floori (x:float2 const) : int2*
- *ceili (x:float2 const) : int2*
- *roundi (x:float2 const) : int2*
- *trunci (x:float2 const) : int2*
- *floori (x:float3 const) : int3*
- *ceili (x:float3 const) : int3*
- *roundi (x:float3 const) : int3*
- *trunci (x:float3 const) : int3*
- *floori (x:float4 const) : int4*
- *ceili (x:float4 const) : int4*
- *roundi (x:float4 const) : int4*
- *trunci (x:float4 const) : int4*
- *- (x:math::float4x4 const implicit) : math::float4x4*
- *- (x:math::float3x4 const implicit) : math::float3x4*
- *- (x:math::float3x3 const implicit) : math::float3x3*

atan_est (*x: float const*)

atan_est returns float

argument	argument type
x	float const

Fast estimation for the *atan*.

atan2_est (*y: float const; x: float const*)

atan2_est returns float

argument	argument type
y	float const
x	float const

returns the fast approximation of arctangent of y/x

atan_est (*x: float2 const*)

atan_est returns float2

argument	argument type
x	float2 const

Fast estimation for the *atan*.

atan2_est (*y: float2 const; x: float2 const*)

atan2_est returns float2

argument	argument type
y	float2 const
x	float2 const

returns the fast approximation of arctangent of y/x

atan_est (*x: float3 const*)

atan_est returns float3

argument	argument type
x	float3 const

Fast estimation for the *atan*.

atan2_est (*y: float3 const; x: float3 const*)

atan2_est returns float3

argument	argument type
y	float3 const
x	float3 const

returns the fast approximation of arctangent of y/x

atan_est (*x: float4 const*)

atan_est returns float4

argument	argument type
x	float4 const

Fast estimation for the *atan*.

atan2_est (*y: float4 const; x: float4 const*)

atan2_est returns float4

argument	argument type
y	float4 const
x	float4 const

returns the fast approximation of arctangent of y/x

rcp_est (*x: float const*)

rcp_est returns float

argument	argument type
x	float const

returns the fast approximation 1/x

rcp_est (*x: float2 const*)

rcp_est returns float2

argument	argument type
x	float2 const

returns the fast approximation 1/x

rcp_est (*x: float3 const*)

rcp_est returns float3

argument	argument type
x	float3 const

returns the fast approximation 1/x

rcp_est (*x: float4 const*)

rcp_est returns float4

argument	argument type
x	float4 const

returns the fast approximation 1/x

fract (*x: float const*)

fract returns float

argument	argument type
x	float const

returns a fraction part of x

rsqrt (*x: float const*)

rsqrt returns float

argument	argument type
x	float const

returns 1/sqrt(x)

rsqrt_est (*x: float const*)

rsqrt_est returns float

argument	argument type
x	float const

returns the fast approximation 1/sqrt(x)

fract (*x: float2 const*)

fract returns float2

argument	argument type
x	float2 const

returns a fraction part of x

rsqrt (*x: float2 const*)

rsqrt returns float2

argument	argument type
x	float2 const

returns 1/sqrt(x)

rsqrt_est (*x: float2 const*)

rsqrt_est returns float2

argument	argument type
x	float2 const

returns the fast approximation 1/sqrt(x)

fract (*x: float3 const*)

fract returns float3

argument	argument type
x	float3 const

returns a fraction part of x

rsqrt (*x: float3 const*)

rsqrt returns float3

argument	argument type
x	float3 const

returns 1/sqrt(x)

rsqrt_est (*x: float3 const*)

rsqrt_est returns float3

argument	argument type
x	float3 const

returns the fast approximation 1/sqrt(x)

fract (*x: float4 const*)

fract returns float4

argument	argument type
x	float4 const

returns a fraction part of x

rsqrt (*x: float4 const*)

rsqrt returns float4

argument	argument type
x	float4 const

returns 1/sqrt(x)

rsqrt_est (*x: float4 const*)

rsqrt_est returns float4

argument	argument type
x	float4 const

returns the fast approximation 1/sqrt(x)

floori (*x: float const*)

floori returns int

argument	argument type
x	float const

returns a integer value representing the largest integer that is less than or equal to x

ceili (*x: float const*)

ceili returns int

argument	argument type
x	float const

returns a value representing the smallest integer (integer type!) that is greater than or equal to arg0

roundi (*x: float const*)

roundi returns int

argument	argument type
x	float const

returns a integer value representing the integer that is closest to x

trunci (*x: float const*)

trunci returns int

argument	argument type
x	float const

returns a integer value representing the float without fraction part of x

floori (*x: double const*)

floori returns int

argument	argument type
x	double const

returns a integer value representing the largest integer that is less than or equal to x

ceili (*x: double const*)

ceili returns int

argument	argument type
x	double const

returns a value representing the smallest integer (integer type!) that is greater than or equal to arg0

roundi (*x: double const*)

roundi returns int

argument	argument type
x	double const

returns a integer value representing the integer that is closest to x

trunci (*x: double const*)

trunci returns int

argument	argument type
x	double const

returns a integer value representing the float without fraction part of x

floori (*x: float2 const*)

floori returns int2

argument	argument type
x	float2 const

returns a integer value representing the largest integer that is less than or equal to x

ceili (*x: float2 const*)

ceili returns int2

argument	argument type
x	float2 const

returns a value representing the smallest integer (integer type!) that is greater than or equal to arg0

roundi (*x: float2 const*)

roundi returns int2

argument	argument type
x	float2 const

returns a integer value representing the integer that is closest to x

trunci (*x: float2 const*)

trunci returns int2

argument	argument type
x	float2 const

returns a integer value representing the float without fraction part of x

floori (*x: float3 const*)

floori returns int3

argument	argument type
x	float3 const

returns a integer value representing the largest integer that is less than or equal to x

ceili (*x: float3 const*)

ceili returns int3

argument	argument type
x	float3 const

returns a value representing the smallest integer (integer type!) that is greater than or equal to arg0

roundi (*x: float3 const*)

roundi returns int3

argument	argument type
x	float3 const

returns a integer value representing the integer that is closest to x

trunci (*x: float3 const*)

trunci returns int3

argument	argument type
x	float3 const

returns a integer value representing the float without fraction part of x

floori (*x: float4 const*)

floori returns int4

argument	argument type
x	float4 const

returns a integer value representing the largest integer that is less than or equal to x

ceili (*x: float4 const*)

ceili returns int4

argument	argument type
x	float4 const

returns a value representing the smallest integer (integer type!) that is greater than or equal to arg0

roundi (*x: float4 const*)

roundi returns int4

argument	argument type
x	float4 const

returns a integer value representing the integer that is closest to x

trunci (*x: float4 const*)

trunci returns int4

argument	argument type
x	float4 const

returns a integer value representing the float without fraction part of x

operator - (*x: float4x4 const implicit*)

- returns *math::float4x4*

argument	argument type
x	*math::float4x4* const implicit

returns -x

operator - (*x: float3x4 const implicit*)

- returns *math::float3x4*

argument	argument type
x	*math::float3x4* const implicit

returns -x

operator - (*x: float3x3 const implicit*)

- returns *math::float3x3*

argument	argument type
x	*math::float3x3* const implicit

returns -x

3.6 float3 only

- *cross (x:float3 const;y:float3 const) : float3*
- *distance (x:float2 const;y:float2 const) : float*
- *distance_sq (x:float2 const;y:float2 const) : float*
- *inv_distance (x:float2 const;y:float2 const) : float*
- *inv_distance_sq (x:float2 const;y:float2 const) : float*
- *distance (x:float3 const;y:float3 const) : float*
- *distance_sq (x:float3 const;y:float3 const) : float*
- *inv_distance (x:float3 const;y:float3 const) : float*
- *inv_distance_sq (x:float3 const;y:float3 const) : float*
- *distance (x:float4 const;y:float4 const) : float*
- *distance_sq (x:float4 const;y:float4 const) : float*
- *inv_distance (x:float4 const;y:float4 const) : float*
- *inv_distance_sq (x:float4 const;y:float4 const) : float*
- *reflect (v:float3 const;n:float3 const) : float3*
- *reflect (v:float2 const;n:float2 const) : float2*
- *refract (v:float3 const;n:float3 const;nint:float const) : float3*
- *refract (v:float2 const;n:float2 const;nint:float const) : float2*

cross (*x: float3 const; y: float3 const*)

cross returns float3

argument	argument type
x	float3 const
y	float3 const

returns vector representing cross product between x and y

distance (*x: float2 const; y: float2 const*)

distance returns float

argument	argument type
x	float2 const
y	float2 const

returns a non-negative value representing distance between x and y

distance_sq (*x: float2 const; y: float2 const*) distance_sq returns float

argument	argument type
x	float2 const
y	float2 const

returns a non-negative value representing squared distance between x and y

inv_distance (*x: float2 const; y: float2 const*)

inv_distance returns float

argument	argument type
x	float2 const
y	float2 const

returns a non-negative value representing 1/distance between x and y

inv_distance_sq (*x: float2 const; y: float2 const*)

inv_distance_sq returns float

argument	argument type
x	float2 const
y	float2 const

returns a non-negative value representing 1/squared distance between x and y

distance (*x: float3 const; y: float3 const*)

distance returns float

argument	argument type
x	float3 const
y	float3 const

returns a non-negative value representing distance between x and y

distance_sq (*x: float3 const; y: float3 const*)

distance_sq returns float

argument	argument type
x	float3 const
y	float3 const

returns a non-negative value representing squared distance between x and y

inv_distance (*x: float3 const; y: float3 const*)

inv_distance returns float

argument	argument type
x	float3 const
y	float3 const

returns a non-negative value representing 1/distance between x and y

inv_distance_sq (*x: float3 const; y: float3 const*)

inv_distance_sq returns float

argument	argument type
x	float3 const
y	float3 const

returns a non-negative value representing 1/squared distance between x and y

distance (*x: float4 const; y: float4 const*)

distance returns float

argument	argument type
x	float4 const
y	float4 const

returns a non-negative value representing distance between x and y

distance_sq (*x: float4 const; y: float4 const*)

distance_sq returns float

argument	argument type
x	float4 const
y	float4 const

returns a non-negative value representing squared distance between x and y

inv_distance (*x: float4 const; y: float4 const*)

inv_distance returns float

argument	argument type
x	float4 const
y	float4 const

returns a non-negative value representing 1/distance between x and y

inv_distance_sq (*x: float4 const; y: float4 const*)

inv_distance_sq returns float

argument	argument type
x	float4 const
y	float4 const

returns a non-negative value representing 1/squared distance between x and y

reflect (*v: float3 const; n: float3 const*)

reflect returns float3

argument	argument type
v	float3 const
n	float3 const

returns vector representing reflection of vector v from normal n same as

```
def reflect(v, n: float3)
    return v - 2. * dot(v, n) * n
```

reflect (*v: float2 const; n: float2 const*)

reflect returns float2

argument	argument type
v	float2 const
n	float2 const

returns vector representing reflection of vector v from normal n same as

```
def reflect(v, n: float3)
    return v - 2. * dot(v, n) * n
```

refract (*v: float3 const; n: float3 const; nint: float const*)

refract returns float3

argument	argument type
v	float3 const
n	float3 const
nint	float const

returns vector representing refractoin of vector v from normal n same as

```
def refract(v, n: float3; nint: float; outRefracted: float3&)
    let dt = dot(v, n)
    let discr = 1. - nint * nint * (1. - dt * dt)
    if discr > 0.
        outRefracted = nint * (v - n * dt) - n * sqrt(discr)
        return true
    return false
```

refract (*v: float2 const; n: float2 const; nint: float const*)

refract returns float2

argument	argument type
v	float2 const
n	float2 const
nint	float const

returns vector representing refractoin of vector v from normal n same as

```
def refract(v, n: float3; nint: float; outRefracted: float3&)
    let dt = dot(v, n)
    let discr = 1. - nint * nint * (1. - dt * dt)
    if discr > 0.
```

(continues on next page)

(continued from previous page)

```
        outRefracted = nint * (v - n * dt) - n * sqrt(discr)
        return true
    return false
```

3.7 float2, float3, float4

- *dot (x:float2 const;y:float2 const) : float*
- *dot (x:float3 const;y:float3 const) : float*
- *dot (x:float4 const;y:float4 const) : float*
- *fast_normalize (x:float2 const) : float2*
- *fast_normalize (x:float3 const) : float3*
- *fast_normalize (x:float4 const) : float4*
- *normalize (x:float2 const) : float2*
- *normalize (x:float3 const) : float3*
- *normalize (x:float4 const) : float4*
- *length (x:float2 const) : float*
- *length (x:float3 const) : float*
- *length (x:float4 const) : float*
- *inv_length (x:float2 const) : float*
- *inv_length (x:float3 const) : float*
- *inv_length (x:float4 const) : float*
- *inv_length_sq (x:float2 const) : float*
- *inv_length_sq (x:float3 const) : float*
- *inv_length_sq (x:float4 const) : float*
- *length_sq (x:float2 const) : float*
- *length_sq (x:float3 const) : float*
- *length_sq (x:float4 const) : float*

dot (*x: float2 const; y: float2 const*)

dot returns float

argument	argument type
x	float2 const
y	float2 const

returns scalar representing dot product between x and y

dot (*x: float3 const; y: float3 const*) dot returns float

argument	argument type
x	float3 const
y	float3 const

returns scalar representing dot product between x and y

dot (*x: float4 const; y: float4 const*)

dot returns float

argument	argument type
x	float4 const
y	float4 const

returns scalar representing dot product between x and y

fast_normalize (*x: float2 const*)

fast_normalize returns float2

argument	argument type
x	float2 const

returns the fast approximation of normalized x, or nan if length(x) is 0

fast_normalize (*x: float3 const*)

fast_normalize returns float3

argument	argument type
x	float3 const

returns the fast approximation of normalized x, or nan if length(x) is 0

fast_normalize (*x: float4 const*)

fast_normalize returns float4

argument	argument type
x	float4 const

returns the fast approximation of normalized x, or nan if length(x) is 0

normalize (*x: float2 const*)

normalize returns float2

argument	argument type
x	float2 const

returns normalized x, or nan if length(x) is 0

normalize (*x: float3 const*)

normalize returns float3

argument	argument type
x	float3 const

returns normalized x, or nan if length(x) is 0

normalize (*x: float4 const*)

normalize returns float4

argument	argument type
x	float4 const

returns normalized x, or nan if length(x) is 0

length (*x: float2 const*)

length returns float

argument	argument type
x	float2 const

returns a non-negative value representing magnitude of x

length (*x: float3 const*)

length returns float

argument	argument type
x	float3 const

returns a non-negative value representing magnitude of x

length (*x: float4 const*)

length returns float

argument	argument type
x	float4 const

returns a non-negative value representing magnitude of x

inv_length (*x: float2 const*)

inv_length returns float

argument	argument type
x	float2 const

returns a non-negative value representing 1/magnitude of x

inv_length (*x: float3 const*)

inv_length returns float

argument	argument type
x	float3 const

returns a non-negative value representing 1/magnitude of x

inv_length (*x: float4 const*)

inv_length returns float

argument	argument type
x	float4 const

returns a non-negative value representing 1/magnitude of x

inv_length_sq (*x: float2 const*)

inv_length_sq returns float

argument	argument type
x	float2 const

returns a non-negative value representing 1/squared magnitude of x

inv_length_sq (*x: float3 const*)

inv_length_sq returns float

argument	argument type
x	float3 const

returns a non-negative value representing 1/squared magnitude of x

inv_length_sq (*x: float4 const*)

inv_length_sq returns float

argument	argument type
x	float4 const

returns a non-negative value representing 1/squared magnitude of x

length_sq (*x: float2 const*)

length_sq returns float

argument	argument type
x	float2 const

returns a non-negative value representing squared magnitude of x

length_sq (*x: float3 const*)

length_sq returns float

argument	argument type
x	float3 const

returns a non-negative value representing squared magnitude of x

length_sq (*x: float4 const*)

length_sq returns float

argument	argument type
x	float4 const

returns a non-negative value representing squared magnitude of x

3.8 Noise functions

- *uint32_hash (seed:uint const) : uint*
- *uint_noise_1D (position:int const;seed:uint const) : uint*
- *uint_noise_2D (position:int2 const;seed:uint const) : uint*
- *uint_noise_3D (position:int3 const;seed:uint const) : uint*

uint32_hash (*seed: uint const*)

uint32_hash returns uint

argument	argument type
seed	uint const

returns hashed value of seed

uint_noise_1D (*position: int const; seed: uint const*)

uint_noise_1D returns uint

argument	argument type
position	int const
seed	uint const

returns noise value of position in the seeded sequence

uint_noise_2D (*position: int2 const; seed: uint const*)

uint_noise_2D returns uint

argument	argument type
position	int2 const
seed	uint const

returns noise value of position in the seeded sequence

uint_noise_3D (*position: int3 const; seed: uint const*)

uint_noise_3D returns uint

argument	argument type
position	int3 const
seed	uint const

returns noise value of position in the seeded sequence

3.9 lerp/mad/clamp

- *mad (a:float const;b:float const;c:float const) : float*
- *lerp (a:float const;b:float const;t:float const) : float*
- *mad (a:float2 const;b:float2 const;c:float2 const) : float2*
- *lerp (a:float2 const;b:float2 const;t:float2 const) : float2*
- *mad (a:float3 const;b:float3 const;c:float3 const) : float3*
- *lerp (a:float3 const;b:float3 const;t:float3 const) : float3*
- *mad (a:float4 const;b:float4 const;c:float4 const) : float4*
- *lerp (a:float4 const;b:float4 const;t:float4 const) : float4*
- *mad (a:float2 const;b:float const;c:float2 const) : float2*
- *mad (a:float3 const;b:float const;c:float3 const) : float3*
- *mad (a:float4 const;b:float const;c:float4 const) : float4*
- *mad (a:int const;b:int const;c:int const) : int*
- *mad (a:int2 const;b:int2 const;c:int2 const) : int2*
- *mad (a:int3 const;b:int3 const;c:int3 const) : int3*
- *mad (a:int4 const;b:int4 const;c:int4 const) : int4*
- *mad (a:int2 const;b:int const;c:int2 const) : int2*
- *mad (a:int3 const;b:int const;c:int3 const) : int3*
- *mad (a:int4 const;b:int const;c:int4 const) : int4*
- *mad (a:uint const;b:uint const;c:uint const) : uint*
- *mad (a:uint2 const;b:uint2 const;c:uint2 const) : uint2*
- *mad (a:uint3 const;b:uint3 const;c:uint3 const) : uint3*
- *mad (a:uint4 const;b:uint4 const;c:uint4 const) : uint4*
- *mad (a:uint2 const;b:uint const;c:uint2 const) : uint2*
- *mad (a:uint3 const;b:uint const;c:uint3 const) : uint3*
- *mad (a:uint4 const;b:uint const;c:uint4 const) : uint4*
- *mad (a:double const;b:double const;c:double const) : double*
- *lerp (a:double const;b:double const;t:double const) : double*
- *clamp (t:int const;a:int const;b:int const) : int*
- *clamp (t:int2 const;a:int2 const;b:int2 const) : int2*
- *clamp (t:int3 const;a:int3 const;b:int3 const) : int3*
- *clamp (t:int4 const;a:int4 const;b:int4 const) : int4*
- *clamp (t:uint const;a:uint const;b:uint const) : uint*

- *clamp (t:uint2 const;a:uint2 const;b:uint2 const) : uint2*
- *clamp (t:uint3 const;a:uint3 const;b:uint3 const) : uint3*
- *clamp (t:uint4 const;a:uint4 const;b:uint4 const) : uint4*
- *clamp (t:float const;a:float const;b:float const) : float*
- *clamp (t:float2 const;a:float2 const;b:float2 const) : float2*
- *clamp (t:float3 const;a:float3 const;b:float3 const) : float3*
- *clamp (t:float4 const;a:float4 const;b:float4 const) : float4*
- *clamp (t:double const;a:double const;b:double const) : double*
- *clamp (t:int64 const;a:int64 const;b:int64 const) : int64*
- *clamp (t:uint64 const;a:uint64 const;b:uint64 const) : uint64*
- *lerp (a:float2 const;b:float2 const;t:float const) : float2*
- *lerp (a:float3 const;b:float3 const;t:float const) : float3*
- *lerp (a:float4 const;b:float4 const;t:float const) : float4*

mad (*a: float const; b: float const; c: float const*)

mad returns float

argument	argument type
a	float const
b	float const
c	float const

returns vector or scalar representing a * b + c

lerp (*a: float const; b: float const; t: float const*)

lerp returns float

argument	argument type
a	float const
b	float const
t	float const

returns vector or scalar representing a + (b - a) * t

mad (*a: float2 const; b: float2 const; c: float2 const*)

mad returns float2

argument	argument type
a	float2 const
b	float2 const
c	float2 const

returns vector or scalar representing a * b + c

lerp (*a: float2 const; b: float2 const; t: float2 const*)

lerp returns float2

argument	argument type
a	float2 const
b	float2 const
t	float2 const

returns vector or scalar representing a + (b - a) * t

mad (*a: float3 const; b: float3 const; c: float3 const*)

mad returns float3

argument	argument type
a	float3 const
b	float3 const
c	float3 const

returns vector or scalar representing a * b + c

lerp (*a: float3 const; b: float3 const; t: float3 const*)

lerp returns float3

argument	argument type
a	float3 const
b	float3 const
t	float3 const

returns vector or scalar representing a + (b - a) * t

mad (*a: float4 const; b: float4 const; c: float4 const*)

mad returns float4

argument	argument type
a	float4 const
b	float4 const
c	float4 const

returns vector or scalar representing a * b + c

lerp (*a: float4 const; b: float4 const; t: float4 const*)

lerp returns float4

argument	argument type
a	float4 const
b	float4 const
t	float4 const

returns vector or scalar representing a + (b - a) * t

mad (*a: float2 const; b: float const; c: float2 const*)

mad returns float2

argument	argument type
a	float2 const
b	float const
c	float2 const

returns vector or scalar representing a * b + c

mad (*a: float3 const; b: float const; c: float3 const*)

mad returns float3

argument	argument type
a	float3 const
b	float const
c	float3 const

returns vector or scalar representing a * b + c

mad (*a: float4 const; b: float const; c: float4 const*)

mad returns float4

argument	argument type
a	float4 const
b	float const
c	float4 const

returns vector or scalar representing a * b + c

mad (*a: int const; b: int const; c: int const*)

mad returns int

argument	argument type
a	int const
b	int const
c	int const

returns vector or scalar representing a * b + c

mad (*a: int2 const; b: int2 const; c: int2 const*)

mad returns int2

argument	argument type
a	int2 const
b	int2 const
c	int2 const

returns vector or scalar representing a * b + c

mad (*a: int3 const; b: int3 const; c: int3 const*)

mad returns int3

argument	argument type
a	int3 const
b	int3 const
c	int3 const

returns vector or scalar representing a * b + c

mad (*a: int4 const; b: int4 const; c: int4 const*)

mad returns int4

argument	argument type
a	int4 const
b	int4 const
c	int4 const

returns vector or scalar representing a * b + c

mad (*a: int2 const; b: int const; c: int2 const*)

mad returns int2

argument	argument type
a	int2 const
b	int const
c	int2 const

returns vector or scalar representing a * b + c

mad (*a: int3 const; b: int const; c: int3 const*)

mad returns int3

argument	argument type
a	int3 const
b	int const
c	int3 const

returns vector or scalar representing a * b + c

mad (*a: int4 const; b: int const; c: int4 const*)

mad returns int4

argument	argument type
a	int4 const
b	int const
c	int4 const

returns vector or scalar representing a * b + c

mad (*a: uint const; b: uint const; c: uint const*)

mad returns uint

argument	argument type
a	uint const
b	uint const
c	uint const

returns vector or scalar representing a * b + c

mad (*a: uint2 const; b: uint2 const; c: uint2 const*)

mad returns uint2

argument	argument type
a	uint2 const
b	uint2 const
c	uint2 const

returns vector or scalar representing a * b + c

mad (*a: uint3 const; b: uint3 const; c: uint3 const*)

mad returns uint3

argument	argument type
a	uint3 const
b	uint3 const
c	uint3 const

returns vector or scalar representing a * b + c

mad (*a: uint4 const; b: uint4 const; c: uint4 const*)

mad returns uint4

argument	argument type
a	uint4 const
b	uint4 const
c	uint4 const

returns vector or scalar representing a * b + c

mad (*a: uint2 const; b: uint const; c: uint2 const*)

mad returns uint2

argument	argument type
a	uint2 const
b	uint const
c	uint2 const

returns vector or scalar representing a * b + c

mad (*a: uint3 const; b: uint const; c: uint3 const*)

mad returns uint3

argument	argument type
a	uint3 const
b	uint const
c	uint3 const

returns vector or scalar representing a * b + c

mad (*a: uint4 const; b: uint const; c: uint4 const*)

mad returns uint4

argument	argument type
a	uint4 const
b	uint const
c	uint4 const

returns vector or scalar representing a * b + c

mad (*a: double const; b: double const; c: double const*)

mad returns double

argument	argument type
a	double const
b	double const
c	double const

returns vector or scalar representing a * b + c

lerp (*a: double const; b: double const; t: double const*)

lerp returns double

argument	argument type
a	double const
b	double const
t	double const

returns vector or scalar representing a + (b - a) * t

clamp (*t: int const; a: int const; b: int const*)

clamp returns int

argument	argument type
t	int const
a	int const
b	int const

returns vector or scalar representing min(max(t, a), b)

clamp (*t: int2 const; a: int2 const; b: int2 const*)

clamp returns int2

argument	argument type
t	int2 const
a	int2 const
b	int2 const

returns vector or scalar representing min(max(t, a), b)

clamp (*t: int3 const; a: int3 const; b: int3 const*)

clamp returns int3

argument	argument type
t	int3 const
a	int3 const
b	int3 const

returns vector or scalar representing min(max(t, a), b)

clamp (*t: int4 const; a: int4 const; b: int4 const*)

clamp returns int4

argument	argument type
t	int4 const
a	int4 const
b	int4 const

returns vector or scalar representing min(max(t, a), b)

clamp (*t: uint const; a: uint const; b: uint const*)

clamp returns uint

argument	argument type
t	uint const
a	uint const
b	uint const

returns vector or scalar representing min(max(t, a), b)

clamp (*t: uint2 const; a: uint2 const; b: uint2 const*)

clamp returns uint2

argument	argument type
t	uint2 const
a	uint2 const
b	uint2 const

returns vector or scalar representing min(max(t, a), b)

clamp (*t: uint3 const; a: uint3 const; b: uint3 const*)

clamp returns uint3

argument	argument type
t	uint3 const
a	uint3 const
b	uint3 const

returns vector or scalar representing min(max(t, a), b)

clamp (*t: uint4 const; a: uint4 const; b: uint4 const*)

clamp returns uint4

argument	argument type
t	uint4 const
a	uint4 const
b	uint4 const

returns vector or scalar representing min(max(t, a), b)

clamp (*t: float const; a: float const; b: float const*)

clamp returns float

argument	argument type
t	float const
a	float const
b	float const

returns vector or scalar representing min(max(t, a), b)

clamp (*t: float2 const; a: float2 const; b: float2 const*)

clamp returns float2

argument	argument type
t	float2 const
a	float2 const
b	float2 const

returns vector or scalar representing min(max(t, a), b)

clamp (*t: float3 const; a: float3 const; b: float3 const*)

clamp returns float3

argument	argument type
t	float3 const
a	float3 const
b	float3 const

returns vector or scalar representing min(max(t, a), b)

clamp (*t: float4 const; a: float4 const; b: float4 const*)

clamp returns float4

argument	argument type
t	float4 const
a	float4 const
b	float4 const

returns vector or scalar representing min(max(t, a), b)

clamp (*t: double const; a: double const; b: double const*)

clamp returns double

argument	argument type
t	double const
a	double const
b	double const

returns vector or scalar representing min(max(t, a), b)

clamp (*t: int64 const; a: int64 const; b: int64 const*)

clamp returns int64

argument	argument type
t	int64 const
a	int64 const
b	int64 const

returns vector or scalar representing min(max(t, a), b)

clamp (*t: uint64 const; a: uint64 const; b: uint64 const*)

clamp returns uint64

argument	argument type
t	uint64 const
a	uint64 const
b	uint64 const

returns vector or scalar representing min(max(t, a), b)

lerp (*a: float2 const; b: float2 const; t: float const*)

lerp returns float2

argument	argument type
a	float2 const
b	float2 const
t	float const

returns vector or scalar representing a + (b - a) * t

lerp (*a: float3 const; b: float3 const; t: float const*)

lerp returns float3

argument	argument type
a	float3 const
b	float3 const
t	float const

returns vector or scalar representing a + (b - a) * t

lerp (*a: float4 const; b: float4 const; t: float const*)

lerp returns float4

argument	argument type
a	float4 const
b	float4 const
t	float const

returns vector or scalar representing a + (b - a) * t

3.10 Matrix operations

- ** (x:math::float4x4 const implicit;y:math::float4x4 const implicit) : math::float4x4*
- *== (x:math::float4x4 const implicit;y:math::float4x4 const implicit) : bool*
- *!= (x:math::float4x4 const implicit;y:math::float4x4 const implicit) : bool*
- ** (x:math::float3x4 const implicit;y:math::float3x4 const implicit) : math::float3x4*
- ** (x:math::float3x4 const implicit;y:float3 const) : float3*
- ** (x:math::float4x4 const implicit;y:float4 const) : float4*
- *== (x:math::float3x4 const implicit;y:math::float3x4 const implicit) : bool*
- *!= (x:math::float3x4 const implicit;y:math::float3x4 const implicit) : bool*
- ** (x:math::float3x3 const implicit;y:math::float3x3 const implicit) : math::float3x3*
- ** (x:math::float3x3 const implicit;y:float3 const) : float3*
- *== (x:math::float3x3 const implicit;y:math::float3x3 const implicit) : bool*
- *!= (x:math::float3x3 const implicit;y:math::float3x3 const implicit) : bool*

operator * (*x: float4x4 const implicit; y: float4x4 const implicit*)

- returns *math::float4x4*

argument	argument type
x	*math::float4x4* const implicit
y	*math::float4x4* const implicit

Multiplies x by y.

operator == (*x: float4x4 const implicit; y: float4x4 const implicit*)

== returns bool

argument	argument type
x	*math::float4x4* const implicit
y	*math::float4x4* const implicit

Compares x and y per component. Returns false if at least one component does not match.

operator != (*x: float4x4 const implicit; y: float4x4 const implicit*)

!= returns bool

argument	argument type
x	*math::float4x4* const implicit
y	*math::float4x4* const implicit

Compares x and y per component. Returns true if at least one component does not match.

operator * (*x: float3x4 const implicit; y: float3x4 const implicit*)

- returns *math::float3x4*

argument	argument type
x	*math::float3x4* const implicit
y	*math::float3x4* const implicit

Multiplies x by y.

operator * (*x: float3x4 const implicit; y: float3 const*)

- returns float3

argument	argument type
x	*math::float3x4* const implicit
y	float3 const

Multiplies x by y.

operator * (*x: float4x4 const implicit; y: float4 const*)

- returns float4

argument	argument type
x	*math::float4x4* const implicit
y	float4 const

Multiplies x by y.

operator == (*x: float3x4 const implicit; y: float3x4 const implicit*)

== returns bool

argument	argument type
x	*math::float3x4* const implicit
y	*math::float3x4* const implicit

Compares x and y per component. Returns false if at least one component does not match.

operator != (*x: float3x4 const implicit; y: float3x4 const implicit*)

!= returns bool

argument	argument type
x	*math::float3x4* const implicit
y	*math::float3x4* const implicit

Compares x and y per component. Returns true if at least one component does not match.

operator * (*x: float3x3 const implicit; y: float3x3 const implicit*)

- returns *math::float3x3*

argument	argument type
x	*math::float3x3* const implicit
y	*math::float3x3* const implicit

Multiplies x by y.

operator * (*x: float3x3 const implicit; y: float3 const*)

- returns float3

argument	argument type
x	*math::float3x3* const implicit
y	float3 const

Multiplies x by y.

operator == (*x: float3x3 const implicit; y: float3x3 const implicit*)

== returns bool

argument	argument type
x	*math::float3x3* const implicit
y	*math::float3x3* const implicit

Compares x and y per component. Returns false if at least one component does not match.

operator != (*x: float3x3 const implicit; y: float3x3 const implicit*)

!= returns bool

argument	argument type
x	*math::float3x3* const implicit
y	*math::float3x3* const implicit

Compares x and y per component. Returns true if at least one component does not match.

3.11 Matrix initializers

- *float3x3 () : math::float3x3*
- *float3x4 () : math::float3x4*
- *float4x4 () : math::float4x4*
- *float4x4 (arg0:math::float3x4 const implicit) : math::float4x4*
- *identity4x4 () : math::float4x4*
- *float3x4 (arg0:math::float4x4 const implicit) : math::float3x4*
- *identity3x4 () : math::float3x4*
- *float3x3 (arg0:math::float4x4 const implicit) : math::float3x3*
- *float3x3 (arg0:math::float3x4 const implicit) : math::float3x3*
- *identity3x3 () : math::float3x3*

float3x3 ()

float3x3 returns *math::float3x3*

Returns empty matrix, where each component is 0.

float3x4 ()

float3x4 returns *math::float3x4*

Returns empty matrix, where each component is 0.

float4x4 ()

float4x4 returns *math::float4x4*

Returns empty matrix, where each component is 0.

float4x4 (*arg0: float3x4 const implicit*)

float4x4 returns *math::float4x4*

argument	argument type
arg0	*math::float3x4* const implicit

Returns empty matrix, where each component is 0.

identity4x4 ()

identity4x4 returns *math::float4x4*

Returns identity matrix, where diagonal is 1 and every other component is 0.

float3x4 (*arg0: float4x4 const implicit*)

float3x4 returns *math::float3x4*

argument	argument type
arg0	*math::float4x4* const implicit

Returns empty matrix, where each component is 0.

identity3x4 ()

identity3x4 returns *math::float3x4*

Returns identity matrix, where diagonal is 1 and every other component is 0.

float3x3 (*arg0: float4x4 const implicit*)

float3x3 returns *math::float3x3*

argument	argument type
arg0	*math::float4x4* const implicit

Returns empty matrix, where each component is 0.

float3x3 (*arg0: float3x4 const implicit*)

float3x3 returns *math::float3x3*

argument	argument type
arg0	*math::float3x4* const implicit

Returns empty matrix, where each component is 0.

identity3x3 ()

identity3x3 returns *math::float3x3*

Returns identity matrix, where diagonal is 1 and every other component is 0.

3.12 Matrix manipulation

- *identity (x:math::float4x4 implicit) : void*
- *translation (xyz:float3 const) : math::float4x4*
- *transpose (x:math::float4x4 const implicit) : math::float4x4*
- *persp_forward (wk:float const;hk:float const;zn:float const;zf:float const) : math::float4x4*
- *persp_reverse (wk:float const;hk:float const;zn:float const;zf:float const) : math::float4x4*
- *look_at (eye:float3 const;at:float3 const;up:float3 const) : math::float4x4*
- *compose (pos:float3 const;rot:float4 const;scale:float3 const) : math::float4x4*
- *decompose (mat:math::float4x4 const implicit;pos:float3& implicit;rot:float4& implicit;scale:float3& implicit) : void*
- *identity (x:math::float3x4 implicit) : void*
- *inverse (x:math::float3x4 const implicit) : math::float3x4*
- *inverse (m:math::float4x4 const implicit) : math::float4x4*
- *orthonormal_inverse (m:math::float3x3 const implicit) : math::float3x3*
- *orthonormal_inverse (m:math::float3x4 const implicit) : math::float3x4*
- *rotate (x:math::float3x4 const implicit;y:float3 const) : float3*
- *identity (x:math::float3x3 implicit) : void*

identity (*x: float4x4 implicit*)

argument	argument type
x	*math::float4x4* implicit

Returns identity matrix, where diagonal is 1 and every other component is 0.

translation (*xyz: float3 const*)

translation returns *math::float4x4*

argument	argument type
xyz	float3 const

produces a translation by xyz

1	0	0	0
0	1	0	0
0	0	1	0
x	y	z	1

transpose (*x: float4x4 const implicit*)

transpose returns *math::float4x4*

argument	argument type
x	*math::float4x4* const implicit

Transposes the specified input matrix x.

persp_forward (*wk: float const; hk: float const; zn: float const; zf: float const*)

persp_forward returns *math::float4x4*

argument	argument type
wk	float const
hk	float const
zn	float const
zf	float const

Perspective matrix, zn - 0, zf - 1

persp_reverse (*wk: float const; hk: float const; zn: float const; zf: float const*)

persp_reverse returns *math::float4x4*

argument	argument type
wk	float const
hk	float const
zn	float const
zf	float const

Perspective matrix, zn - 1, zf - 0

look_at (*eye: float3 const; at: float3 const; up: float3 const*)

look_at returns *math::float4x4*

argument	argument type
eye	float3 const
at	float3 const
up	float3 const

Look-at matrix with the origin at *eye*, looking at *at*, with *up* as up direction.

compose (*pos: float3 const; rot: float4 const; scale: float3 const*)

compose returns *math::float4x4*

argument	argument type
pos	float3 const
rot	float4 const
scale	float3 const

Compose transformation out of translation, rotation and scale.

decompose (*mat: float4x4 const implicit; pos: float3& implicit; rot: float4& implicit; scale: float3& implicit*)

argument	argument type
mat	*math::float4x4* const implicit
pos	float3& implicit
rot	float4& implicit
scale	float3& implicit

Decompose transformation into translation, rotation and scale.

identity (*x: float3x4 implicit*)

argument	argument type
x	*math::float3x4* implicit

Returns identity matrix, where diagonal is 1 and every other component is 0.

inverse (*x: float3x4 const implicit*)

inverse returns *math::float3x4*

argument	argument type
x	*math::float3x4* const implicit

Returns the inverse of the matrix x.

inverse (*m: float4x4 const implicit*)

inverse returns *math::float4x4*

argument	argument type
m	*math::float4x4* const implicit

Returns the inverse of the matrix x.

orthonormal_inverse (*m: float3x3 const implicit*)

orthonormal_inverse returns *math::float3x3*

argument	argument type
m	*math::float3x3* const implicit

Fast *inverse* for the orthonormal matrix.

orthonormal_inverse (*m: float3x4 const implicit*)

orthonormal_inverse returns *math::float3x4*

argument	argument type
m	*math::float3x4* const implicit

Fast *inverse* for the orthonormal matrix.

rotate (*x: float3x4 const implicit; y: float3 const*)

rotate returns float3

argument	argument type
x	*math::float3x4* const implicit
y	float3 const

Rotates vector y by 3x4 matrix x. Only 3x3 portion of x is multiplied by y.

identity (*x: float3x3 implicit*)

argument	argument type
x	*math::float3x3* implicit

Returns identity matrix, where diagonal is 1 and every other component is 0.

3.13 Quaternion operations

- *quat_from_unit_arc (v0:float3 const;v1:float3 const) : float4*
- *quat_from_unit_vec_ang (v:float3 const;ang:float const) : float4*
- *un_quat (m:math::float4x4 const implicit) : float4*
- *quat_mul (q1:float4 const;q2:float4 const) : float4*
- *quat_mul_vec (q:float4 const;v:float3 const) : float3*
- *quat_conjugate (q:float4 const) : float4*

quat_from_unit_arc (*v0: float3 const; v1: float3 const*)

quat_from_unit_arc returns float4

argument	argument type
v0	float3 const
v1	float3 const

Quaternion which represents rotation from *v0* to *v1*, both arguments need to be normalized

quat_from_unit_vec_ang (*v: float3 const; ang: float const*)

quat_from_unit_vec_ang returns float4

argument	argument type
v	float3 const
ang	float const

Quaternion which represents rotation for *ang* radians around vector *v*. *v* needs to be normalized

un_quat (*m: float4x4 const implicit*)

un_quat returns float4

argument	argument type
m	*math::float4x4* const implicit

Quaternion from the rotation part of the matrix

quat_mul (*q1: float4 const; q2: float4 const*)

quat_mul returns float4

argument	argument type
q1	float4 const
q2	float4 const

Quaternion which is multiplication of *q1* and *q2*

quat_mul_vec (*q: float4 const; v: float3 const*)

quat_mul_vec returns float3

argument	argument type
q	float4 const
v	float3 const

Transform vector *v* by quaternion *q*

quat_conjugate (*q: float4 const*)

quat_conjugate returns float4

argument	argument type
q	float4 const

Quaternion which is conjugate of *q*

3.14 Packing and unpacking

- *pack_float_to_byte (x:float4 const) : uint*
- *unpack_byte_to_float (x:uint const) : float4*

pack_float_to_byte (*x: float4 const*)

pack_float_to_byte returns uint

argument	argument type
x	float4 const

Packs float4 vector *v* to byte4 vector and returns it as uint. Each component is clamped to [0..255] range.

unpack_byte_to_float (*x: uint const*)

unpack_byte_to_float returns float4

argument	argument type
x	uint const

Unpacks byte4 vector to float4 vector.

3.15 Uncategorized

operator [] (*m: float4x4 implicit ==const; i: int const*)

[] returns float4&

argument	argument type
m	*math::float4x4* implicit!
i	int const

Returns the component of the matrix *m* at the specified row.

operator [] (*m: float4x4 const implicit ==const; i: int const*)

[] returns float4 const&

argument	argument type
m	*math::float4x4* const implicit!
i	int const

Returns the component of the matrix *m* at the specified row.

operator [] (*m: float4x4 implicit ==const; i: uint const*)

[] returns float4&

argument	argument type
m	*math::float4x4* implicit!
i	uint const

Returns the component of the matrix *m* at the specified row.

operator [] (*m: float4x4 const implicit ==const; i: uint const*)

[] returns float4 const&

argument	argument type
m	*math::float4x4* const implicit!
i	uint const

Returns the component of the matrix *m* at the specified row.

determinant (*x: float4x4 const implicit*) determinant returns float

argument	argument type
x	*math::float4x4* const implicit

Returns the determinant of the matrix *m*.

determinant (*x: float3x4 const implicit*)

determinant returns float

argument	argument type
x	*math::float3x4* const implicit

Returns the determinant of the matrix *m*.

operator [] (*m: float3x4 implicit ==const; i: int const*)

[] returns float3&

argument	argument type
m	*math::float3x4* implicit!
i	int const

Returns the component of the matrix *m* at the specified row.

operator [] (*m: float3x4 const implicit ==const; i: int const*)

[] returns float3 const&

argument	argument type
m	*math::float3x4* const implicit!
i	int const

Returns the component of the matrix *m* at the specified row.

operator [] (*m: float3x4 implicit ==const; i: uint const*)

[] returns float3&

argument	argument type
m	*math::float3x4* implicit!
i	uint const

Returns the component of the matrix *m* at the specified row.

operator [] (*m: float3x4 const implicit ==const; i: uint const*)

[] returns float3 const&

argument	argument type
m	*math::float3x4* const implicit!
i	uint const

Returns the component of the matrix *m* at the specified row.

quat_from_euler (*angles: float3 const*)

quat_from_euler returns float4

argument	argument type
angles	float3 const

Construct quaternion from euler angles.

quat_from_euler (*x: float const; y: float const; z: float const*)

quat_from_euler returns float4

argument	argument type
x	float const
y	float const
z	float const

Construct quaternion from euler angles.

euler_from_un_quat (*angles: float4 const*)

euler_from_un_quat returns float3

argument	argument type
angles	float4 const

Construct euler angles from quaternion.

determinant (*x: float3x3 const implicit*)

determinant returns float

argument	argument type
x	*math::float3x3* const implicit

Returns the determinant of the matrix *m*.

operator [] (*m: float3x3 implicit ==const; i: int const*)

[] returns float3&

argument	argument type
m	*math::float3x3* implicit!
i	int const

Returns the component of the matrix *m* at the specified row.

operator [] (*m: float3x3 const implicit ==const; i: int const*)

[] returns float3 const&

argument	argument type
m	*math::float3x3* const implicit!
i	int const

Returns the component of the matrix *m* at the specified row.

operator [] (*m: float3x3 implicit ==const; i: uint const*)

[] returns float3&

argument	argument type
m	*math::float3x3* implicit!
i	uint const

Returns the component of the matrix *m* at the specified row.

operator [] (*m: float3x3 const implicit ==const; i: uint const*)

[] returns float3 const&

argument	argument type
m	*math::float3x3* const implicit!
i	uint const

Returns the component of the matrix *m* at the specified row.

CHAPTER

FOUR

MATH BIT HELPERS

This module represents collection of bit representation routines, which allow accessing integer and floating point values packed into different types.

All functions and symbols are in “math_bits” module, or publicly available via “math_boost”. Use require to get access to it.

```
require daslib/math_bits
require daslib/math_boost
```

4.1 Type aliases

Vec4f is a variant type

data	float4
i64	int64
i32	int
i16	int16
i8	int8
str	string
ptr	void?
b	bool

bit-castable float4

4.2 float in int,uint

- *int_bits_to_float (x:int const) : float*
- *int_bits_to_float (x:int2 const) : float2*
- *int_bits_to_float (x:int3 const) : float3*
- *int_bits_to_float (x:int4 const) : float4*
- *uint_bits_to_float (x:uint const) : float*
- *uint_bits_to_float (x:uint2 const) : float2*
- *uint_bits_to_float (x:uint3 const) : float3*
- *uint_bits_to_float (x:uint4 const) : float4*

int_bits_to_float (*x: int const*)

int_bits_to_float returns float

argument	argument type
x	int const

bit representation of x is interpreted as a float

int_bits_to_float (*x: int2 const*)

int_bits_to_float returns float2

argument	argument type
x	int2 const

bit representation of x is interpreted as a float

int_bits_to_float (*x: int3 const*)

int_bits_to_float returns float3

argument	argument type
x	int3 const

bit representation of x is interpreted as a float

int_bits_to_float (*x: int4 const*)

int_bits_to_float returns float4

argument	argument type
x	int4 const

bit representation of x is interpreted as a float

uint_bits_to_float (*x: uint const*)

uint_bits_to_float returns float

argument	argument type
x	uint const

bit representation of x is interpreted as a float

uint_bits_to_float (*x: uint2 const*)

uint_bits_to_float returns float2

argument	argument type
x	uint2 const

bit representation of x is interpreted as a float

uint_bits_to_float (*x: uint3 const*)

uint_bits_to_float returns float3

argument	argument type
x	uint3 const

bit representation of x is interpreted as a float

uint_bits_to_float (*x: uint4 const*)

uint_bits_to_float returns float4

argument	argument type
x	uint4 const

bit representation of x is interpreted as a float

4.3 int,uint in float

- *float_bits_to_int (x:float const) : int*
- *float_bits_to_int (x:float2 const) : int2*
- *float_bits_to_int (x:float3 const) : int3*
- *float_bits_to_int (x:float4 const) : int4*
- *float_bits_to_uint (x:float const) : uint*
- *float_bits_to_uint (x:float2 const) : uint2*
- *float_bits_to_uint (x:float3 const) : uint3*
- *float_bits_to_uint (x:float4 const) : uint4*

float_bits_to_int (*x: float const*)

float_bits_to_int returns int

argument	argument type
x	float const

bit representation of x is interpreted as a int

float_bits_to_int (*x: float2 const*)

float_bits_to_int returns int2

argument	argument type
x	float2 const

bit representation of x is interpreted as a int

float_bits_to_int (*x: float3 const*)

float_bits_to_int returns int3

argument	argument type
x	float3 const

bit representation of x is interpreted as a int

float_bits_to_int (*x: float4 const*)

float_bits_to_int returns int4

argument	argument type
x	float4 const

bit representation of x is interpreted as a int

float_bits_to_uint (*x: float const*)

float_bits_to_uint returns uint

argument	argument type
x	float const

bit representation of x is interpreted as a uint

float_bits_to_uint (*x: float2 const*)

float_bits_to_uint returns uint2

argument	argument type
x	float2 const

bit representation of x is interpreted as a uint

float_bits_to_uint (*x: float3 const*)

float_bits_to_uint returns uint3

argument	argument type
x	float3 const

bit representation of x is interpreted as a uint

float_bits_to_uint (*x: float4 const*)

float_bits_to_uint returns uint4

argument	argument type
x	float4 const

bit representation of x is interpreted as a uint

4.4 int64,uint64 in double

- *int64_bits_to_double (x:int64 const) : double*
- *uint64_bits_to_double (x:uint64 const) : double*
- *double_bits_to_int64 (x:double const) : int64*
- *double_bits_to_uint64 (x:double const) : uint64*

int64_bits_to_double (*x: int64 const*)

int64_bits_to_double returns double

argument	argument type
x	int64 const

bit representation of x is interpreted as a double

uint64_bits_to_double (*x: uint64 const*)

uint64_bits_to_double returns double

argument	argument type
x	uint64 const

bit representation of x is interpreted as a double

double_bits_to_int64 (*x: double const*)

double_bits_to_int64 returns int64

argument	argument type
x	double const

bit representation of x is interpreted as a int64

double_bits_to_uint64 (*x: double const*)

double_bits_to_uint64 returns uint64

argument	argument type
x	double const

bit representation of x is interpreted as a uint64

4.5 bit-cast vec4f

- *cast_to_vec4f (x:bool const) : float4*
- *cast_to_vec4f (x:int64 const) : float4*
- *cast_to_int64 (data:float4 const) : int64*
- *cast_to_int32 (data:float4 const) : int*
- *cast_to_int16 (data:float4 const) : int16*
- *cast_to_int8 (data:float4 const) : int8*
- *cast_to_string (data:float4 const) : string*
- *cast_to_pointer (data:float4 const) : void?*

cast_to_vec4f (*x: bool const*)

cast_to_vec4f returns float4

argument	argument type
x	bool const

return a float4 which stores bit-cast version of x

cast_to_vec4f (*x: int64 const*)

cast_to_vec4f returns float4

argument	argument type
x	int64 const

return a float4 which stores bit-cast version of x

cast_to_int64 (*data: float4 const*)

cast_to_int64 returns int64

argument	argument type
data	float4 const

return an int64 which was bit-cast from x

cast_to_int32 (*data: float4 const*)

cast_to_int32 returns int

argument	argument type
data	float4 const

return an int32 which was bit-cast from x

cast_to_int16 (*data: float4 const*)

cast_to_int16 returns int16

argument	argument type
data	float4 const

return an int16 which was bit-cast from x

cast_to_int8 (*data: float4 const*)

cast_to_int8 returns int8

argument	argument type
data	float4 const

return an int8 which was bit-cast from x

cast_to_string (*data: float4 const*)

cast_to_string returns string

argument	argument type
data	float4 const

return a string which pointer was bit-cast from x

cast_to_pointer (*data: float4 const*)

cast_to_pointer returns void?

argument	argument type
data	float4 const

return a pointer which was bit-cast from x

CHAPTER

FIVE

BOOST PACKAGE FOR MATH

The math boost module implements collection of helper macros and functions to accompany *math*.

All functions and symbols are in "math_boost" module, use require to get access to it.

```
require daslib/math_boost
```

AABR

AABR fields are

min	float2
max	float2

axis aligned bounding rectangle

AABB

AABB fields are

min	float3
max	float3

axis aligned bounding box

Ray

Ray fields are

dir	float3
origin	float3

ray (direction and origin)

5.1 Angle conversions

- *degrees (f:float const) : float*
- *radians (f:float const) : float*

degrees (*f: float const*)

degrees returns float

argument	argument type
f	float const

convert radians to degrees

radians (*f: float const*)

radians returns float

argument	argument type
f	float const

convert degrees to radians

5.2 Intersections

- *is_intersecting (a:math_boost::AABR const;b:math_boost::AABR const) : bool*
- *is_intersecting (a:math_boost::AABB const;b:math_boost::AABB const) : bool*
- *is_intersecting (ray:math_boost::Ray const;aabb:math_boost::AABB const;Tmin:float const;Tmax:float const) : bool*

is_intersecting (*a: AABR const; b: AABR const*)

is_intersecting returns bool

argument	argument type
a	*math_boost::AABR* const
b	*math_boost::AABR* const

returns true if inputs intersect

is_intersecting (*a: AABB const; b: AABB const*)

is_intersecting returns bool

argument	argument type
a	*math_boost::AABB* const
b	*math_boost::AABB* const

returns true if inputs intersect

is_intersecting(*ray: Ray const; aabb: AABB const; Tmin: float const; Tmax: float const*)

is_intersecting returns bool

argument	argument type
ray	*math_boost::Ray* const
aabb	*math_boost::AABB* const
Tmin	float const
Tmax	float const

returns true if inputs intersect

5.3 Matrices

- *look_at_lh (Eye:float3 const;At:float3 const;Up:float3 const) : math::float4x4*
- *look_at_rh (Eye:float3 const;At:float3 const;Up:float3 const) : math::float4x4*
- *perspective_lh (fovy:float const;aspect:float const;zn:float const;zf:float const) : math::float4x4*
- *perspective_rh (fovy:float const;aspect:float const;zn:float const;zf:float const) : math::float4x4*
- *perspective_rh_opengl (fovy:float const;aspect:float const;zn:float const;zf:float const) : math::float4x4*
- *ortho_rh (left:float const;right:float const;bottom:float const;top:float const;zNear:float const;zFar:float const) : math::float4x4*
- *planar_shadow (Light:float4 const;Plane:float4 const) : math::float4x4*

look_at_lh(*Eye: float3 const; At: float3 const; Up: float3 const*)

look_at_lh returns *math::float4x4*

argument	argument type
Eye	float3 const
At	float3 const
Up	float3 const

left-handed (z forward) look at matrix with origin at *Eye* and target at *At*, and up vector *Up*.

look_at_rh (*Eye: float3 const; At: float3 const; Up: float3 const*)

look_at_rh returns *math::float4x4*

argument	argument type
Eye	float3 const
At	float3 const
Up	float3 const

right-handed (z towards viewer) look at matrix with origin at *Eye* and target at *At*, and up vector *Up*.

perspective_lh (*fovy: float const; aspect: float const; zn: float const; zf: float const*)

perspective_lh returns *math::float4x4*

argument	argument type
fovy	float const
aspect	float const
zn	float const
zf	float const

left-handed (z forward) perspective matrix

perspective_rh (*fovy: float const; aspect: float const; zn: float const; zf: float const*)

perspective_rh returns *math::float4x4*

argument	argument type
fovy	float const
aspect	float const
zn	float const
zf	float const

right-handed (z toward viewer) perspective matrix

perspective_rh_opengl (*fovy: float const; aspect: float const; zn: float const; zf: float const*)

perspective_rh_opengl returns *math::float4x4*

argument	argument type
fovy	float const
aspect	float const
zn	float const
zf	float const

right-handed (z toward viewer) opengl (z in [-1..1]) perspective matrix

ortho_rh (*left: float const; right: float const; bottom: float const; top: float const; zNear: float const; zFar: float const*)

ortho_rh returns *math::float4x4*

argument	argument type
left	float const
right	float const
bottom	float const
top	float const
zNear	float const
zFar	float const

right handed (z towards viwer) orthographic (parallel) projection matrix

planar_shadow (*Light: float4 const; Plane: float4 const*)

planar_shadow returns *math::float4x4*

argument	argument type
Light	float4 const
Plane	float4 const

planar shadow projection matrix, i.e. all light shadows to be projected on a plane

5.4 Plane

- *plane_dot (Plane:float4 const;Vec:float4 const) : float*
- *plane_normalize (Plane:float4 const) : float4 const*
- *plane_from_point_normal (p:float3 const;n:float3 const) : float4*

plane_dot (*Plane: float4 const; Vec: float4 const*)

plane_dot returns float

argument	argument type
Plane	float4 const
Vec	float4 const

dot product of *Plane* and 'Vec'

plane_normalize (*Plane: float4 const*)

plane_normalize returns float4 const

argument	argument type
Plane	float4 const

normalize `Plane', length xyz will be 1.0 (or 0.0 for no plane)

plane_from_point_normal (*p: float3 const; n: float3 const*)

plane_from_point_normal returns float4

argument	argument type
p	float3 const
n	float3 const

construct plane from point *p* and normal *n*

5.5 Color packig and unpacking

- *RGBA_TO_UCOLOR (x:float const;y:float const;z:float const;w:float const) : uint*
- *RGBA_TO_UCOLOR (xyzw:float4 const) : uint*
- *UCOLOR_TO_RGBA (x:uint const) : float4*
- *UCOLOR_TO_RGB (x:uint const) : float3*

RGBA_TO_UCOLOR (*x: float const; y: float const; z: float const; w: float const*)

RGBA_TO_UCOLOR returns uint

argument	argument type
x	float const
y	float const
z	float const
w	float const

conversion from RGBA to ucolor. x,y,z,w are in [0,1] range

RGBA_TO_UCOLOR (*xyzw: float4 const*)

RGBA_TO_UCOLOR returns uint

argument	argument type
xyzw	float4 const

conversion from RGBA to ucolor. x,y,z,w are in [0,1] range

UCOLOR_TO_RGBA (*x: uint const*)

UCOLOR_TO_RGBA returns float4

argument	argument type
x	uint const

conversion from ucolor to RGBA. x components are in [0,255] range

UCOLOR_TO_RGB (*x: uint const*)

UCOLOR_TO_RGB returns float3

argument	argument type
x	uint const

conversion from ucolor to RGB. x components are in [0,255] range. result is float3(x,y,z)

5.6 Uncategorized

linear_to_SRGB (*x: float const*)

linear_to_SRGB returns float

argument	argument type
x	float const

convert value from linear space to sRGB curve space

linear_to_SRGB (*c: float3 const*)

linear_to_SRGB returns float3

argument	argument type
c	float3 const

convert value from linear space to sRGB curve space

linear_to_SRGB (*c: float4 const*)

linear_to_SRGB returns float4

argument	argument type
c	float4 const

convert value from linear space to sRGB curve space

CHAPTER

SIX

FILE INPUT OUTPUT LIBRARY

The FIO module exposes C++ FILE * API, file mapping, directory and file stat manipulation routines to Daslang.

All functions and symbols are in “fio” module, use require to get access to it.

```
require fio
```

6.1 Type aliases

file = fio::FILE const?

alias for the *FILE const?*; its there since most file functions expect exactly this type

6.2 Constants

seek_set = 0

constant for *fseek* which sets the file pointer to the beginning of the file plus the offset.

seek_cur = 1

constant for *fseek* which sets the file pointer to the current position of the file plus the offset.

seek_end = 2

constant for *fseek* which sets the file pointer to the end of the file plus the offset.

df_magic = 0x12345678

obsolete. magic number for *binary_save* and *binary_load*.

df_header

df_header fields are

magic	uint
size	int

obsolete. header for the *fsave* and *fload* which internally use *binary_save* and *binary_load*.

6.3 Handled structures

FStat

FStat fields are

is_valid	bool

FStat property operators are

size	uint64
atime	*builtin::clock*
ctime	*builtin::clock*
mtime	*builtin::clock*
is_reg	bool
is_dir	bool

stat and *fstat* return file information in this structure.

6.4 Handled types

FILE

Holds system specific *FILE* type.

6.5 File manipulation

- *remove (name:string const implicit) : bool*
- *fopen (name:string const implicit;mode:string const implicit) : fio::FILE const? const*
- *fclose (file:fio::FILE const? const implicit;context:__context const;line:__lineInfo const) : void*
- *fflush (file:fio::FILE const? const implicit;context:__context const;line:__lineInfo const) : void*
- *fprint (file:fio::FILE const? const implicit;text:string const implicit;context:__context const;line:__lineInfo const) : void*
- *fread (file:fio::FILE const? const implicit;context:__context const;line:__lineInfo const) : string*
- *fmap (file:fio::FILE const? const implicit;block:block<(var arg0:array<uint8>#):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *fgets (file:fio::FILE const? const implicit;context:__context const;line:__lineInfo const) : string*
- *fwrite (file:fio::FILE const? const implicit;text:string const implicit;context:__context const;line:__lineInfo const) : void*

- *feof (file:fio::FILE const? const implicit) : bool*
- *fseek (file:fio::FILE const? const implicit;offset:int64 const;mode:int const;context:__context const;line:__lineInfo const) : int64*
- *ftell (file:fio::FILE const? const implicit;context:__context const;line:__lineInfo const) : int64*
- *fstat (file:fio::FILE const? const implicit;stat:fio::FStat implicit;context:__context const;line:__lineInfo const) : bool*
- *stat (file:string const implicit;stat:fio::FStat implicit) : bool*
- *fstdin () : fio::FILE const? const*
- *fstdout () : fio::FILE const? const*
- *fstderr () : fio::FILE const? const*
- *getchar () : int*
- *fload (file:fio::FILE const? const;size:int const;blk:block<(data:array<uint8> const):void> const) : void*
- *fopen (name:string const;mode:string const;blk:block<(f:fio::FILE const? const):void> const) : auto*
- *stat (path:string const) : fio::FStat*
- *fstat (f:fio::FILE const? const) : fio::FStat*
- *fread (f:fio::FILE const? const;blk:block<(data:string const#):auto> const) : auto*
- *fload (f:fio::FILE const? const;buf:auto(BufType) const -const) : auto*
- *fsave (f:fio::FILE const? const;buf:auto(BufType) const) : auto*
- *fread (f:fio::FILE const? const;buf:auto(BufType) const implicit) : auto*
- *fread (f:fio::FILE const? const;buf:array<auto(BufType)> const implicit) : auto*
- *fwrite (f:fio::FILE const? const;buf:auto(BufType) const implicit) : auto*
- *fwrite (f:fio::FILE const? const;buf:array<auto(BufType)> const implicit) : auto*

remove (*name: string const implicit*)

remove returns bool

argument	argument type
name	string const implicit

deletes file specified by name

fopen (*name: string const implicit; mode: string const implicit*)

fopen returns *fio::FILE* const? const

argument	argument type
name	string const implicit
mode	string const implicit

equivalent to C *fopen*. Opens file in different modes.

fclose (*file: fio::FILE const? const implicit*)

argument	argument type
file	*fio::FILE* const? const implicit

equivalent to C *fclose*. Closes file.

fflush (*file: fio::FILE const? const implicit*)

argument	argument type
file	*fio::FILE* const? const implicit

equivalent to C *fflush*. Flushes FILE buffers.

fprint (*file: fio::FILE const? const implicit; text: string const implicit*)

argument	argument type
file	*fio::FILE* const? const implicit
text	string const implicit

same as *print* but outputs to file.

fread (*file: fio::FILE const? const implicit*)

fread returns string

argument	argument type
file	*fio::FILE* const? const implicit

reads data from file.

fmap (*file: fio::FILE const? const implicit; block: block<(var arg0:array<uint8>#):void> const implicit*)

argument	argument type
file	*fio::FILE* const? const implicit
block	block<(array<uint8>#):void> const implicit

create map view of file, i.e. maps file contents to memory. Data is available as array<uint8> inside the block.

fgets (*file: fio::FILE const? const implicit*) fgets returns string

argument	argument type
file	*fio::FILE* const? const implicit

equivalent to C *fgets*. Reads and returns new string from the line.

fwrite (*file: fio::FILE const? const implicit; text: string const implicit*)

argument	argument type
file	*fio::FILE* const? const implicit
text	string const implicit

writes data fo file.

feof (*file: fio::FILE const? const implicit*)

feof returns bool

argument	argument type
file	*fio::FILE* const? const implicit

equivalent to C *feof*. Returns true if end of file has been reached.

fseek (*file: fio::FILE const? const implicit; offset: int64 const; mode: int const*)

fseek returns int64

argument	argument type
file	*fio::FILE* const? const implicit
offset	int64 const
mode	int const

equivalent to C *fseek*. Rewinds position of the current FILE pointer.

ftell (*file: fio::FILE const? const implicit*)

ftell returns int64

argument	argument type
file	*fio::FILE* const? const implicit

equivalent to C *ftell*. Returns current FILE pointer position.

fstat (*file: fio::FILE const? const implicit; stat: FStat implicit*)

fstat returns bool

argument	argument type
file	*fio::FILE* const? const implicit
stat	*fio::FStat* implicit

equivalent to C *fstat*. Returns information about file, such as file size, timestamp, etc.

stat (*file: string const implicit; stat: FStat implicit*)

stat returns bool

argument	argument type
file	string const implicit
stat	*fio::FStat* implicit

same as fstat, but file is specified by file name.

fstdin ()

fstdin returns *fio::FILE* const? const

returns FILE pointer to standard input.

fstdout ()

fstdout returns *fio::FILE* const? const

returns FILE pointer to standard output.

fstderr ()

fstderr returns *fio::FILE* const? const

returns FILE pointer to standard error.

getchar ()

getchar returns int

equivalent to C *getchar*. Reads and returns next character from standard input.

fload (*file: file; size: int const; blk: block<(data:array<uint8> const):void> const*)

argument	argument type
file	*file*
size	int const
blk	block<(data:array<uint8> const):void> const

obsolete. saves data to file.

fopen (*name: string const; mode: string const; blk: block<(f:fio::FILE const? const):void> const*)

fopen returns auto

argument	argument type
name	string const
mode	string const
blk	block<(f: *file*):void> const

equivalent to C *fopen*. Opens file in different modes.

stat (*path: string const*)

stat returns *fio::FStat*

argument	argument type
path	string const

same as fstat, but file is specified by file name.

fstat (*f: file*)

fstat returns *fio::FStat*

argument	argument type
f	*file*

equivalent to C *fstat*. Returns information about file, such as file size, timestamp, etc.

fread (*f: file; blk: block<(data:string const#):auto> const*)

fread returns auto

argument	argument type
f	*file*
blk	block<(data:string const#):auto> const

reads data from file.

fload (*f: file; buf: auto(BufType) const*)

fload returns auto

argument	argument type
f	*file*
buf	auto(BufType) const

obsolete. saves data to file.

fsave (*f: file; buf: auto(BufType) const*)

fsave returns auto

argument	argument type
f	*file*
buf	auto(BufType) const

obsolete. loads data from file.

fread (*f: file; buf: auto(BufType) const implicit*)

fread returns auto

argument	argument type
f	*file*
buf	auto(BufType) const implicit

reads data from file.

fread (*f: file; buf: array<auto(BufType)> const implicit*)

fread returns auto

argument	argument type
f	*file*
buf	array<auto(BufType)> const implicit

reads data from file.

fwrite (*f: file; buf: auto(BufType) const implicit*)

fwrite returns auto

argument	argument type
f	*file*
buf	auto(BufType) const implicit

writes data fo file.

fwrite (*f: file; buf: array<auto(BufType)> const implicit*)

fwrite returns auto

argument	argument type
f	*file*
buf	array<auto(BufType)> const implicit

writes data fo file.

6.6 Path manipulation

- *dir_name (name:string const implicit;context:__context const;line:__lineInfo const) : string*
- *base_name (name:string const implicit;context:__context const;line:__lineInfo const) : string*
- *get_full_file_name (path:string const implicit;context:__context const;at:__lineInfo const) : string*

dir_name (*name: string const implicit*)

dir_name returns string

argument	argument type
name	string const implicit

equivalent to linux *dirname*. Splits path and returns the component preceding the final '/'. Trailing '/' characters are not counted as part of the pathname.

base_name (*name: string const implicit*)

base_name returns string

argument	argument type
name	string const implicit

equivalent to linux *basename*. Splits path and returns the string up to, but not including, the final '/'.

get_full_file_name (*path: string const implicit*)

get_full_file_name returns string

argument	argument type
path	string const implicit

returns full name of the file in normalized form.

6.7 Directory manipulation

- *mkdir (path:string const implicit) : bool*
- *dir (path:string const;blk:block<(filename:string const):void> const) : auto*

mkdir (*path: string const implicit*)

mkdir returns bool

argument	argument type
path	string const implicit

makes directory.

dir (*path: string const; blk: block<(filename:string const):void> const*)

dir returns auto

argument	argument type
path	string const
blk	block<(filename:string const):void> const

iterates through all files in the specified *path*.

6.8 OS specific routines

- *sleep (msec:uint const) : void*
- *exit (exitCode:int const) : void*
- *popen (command:string const implicit;scope:block<(arg0:fio::FILE const? const):void> const implicit;context:__context const;at:__lineInfo const) : int*
- *popen_binary (command:string const implicit;scope:block<(arg0:fio::FILE const? const):void> const implicit;context:__context const;at:__lineInfo const) : int*
- *get_env_variable (var:string const implicit;context:__context const) : string*
- *sanitize_command_line (var:string const implicit;context:__context const;at:__lineInfo const) : string*

sleep (*msec: uint const*)

argument	argument type
msec	uint const

sleeps for specified number of milliseconds.

exit (*exitCode: int const*)

> **Warning:** This is unsafe operation.

argument	argument type
exitCode	int const

equivalent to C *exit*. Terminates program.

popen (*command: string const implicit; scope: block<(arg0:fio::FILE const? const):void> const implicit*)

popen returns int

> **Warning:** This is unsafe operation.

argument	argument type
command	string const implicit
scope	block<(*fio::FILE* const? const):void> const implicit

equivalent to linux *popen*. Opens pipe to command.

popen_binary (*command: string const implicit; scope: block<(arg0:fio::FILE const? const):void> const implicit*) popen_binary returns int

Warning: This is unsafe operation.

argument	argument type
command	string const implicit
scope	block<(*fio::FILE* const? const):void> const implicit

opens pipe to command and returns FILE pointer to it, in binary mode.

`get_env_variable` (*var: string const implicit*)

get_env_variable returns string

argument	argument type
var	string const implicit

returns value of the environment variable.

`sanitize_command_line` (*var: string const implicit*)

sanitize_command_line returns string

argument	argument type
var	string const implicit

sanitizes command line arguments.

CHAPTER

SEVEN

RANDOM GENERATOR LIBRARY

The random library implements basic random routines.

All functions and symbols are in “random” module, use require to get access to it.

```
require random
```

7.1 Constants

LCG_RAND_MAX = 32767

maximum possible output of random number generator

LCG_RAND_MAX_BIG = 1073741823

maximum possible output of random_big_int

7.2 Seed and basic generators

- *random_seed (seed:int const) : auto*
- *random_seed2D (seed:int4& -const;co:int2 const;cf:int const) : auto*
- *random_int (seed:int4& -const) : auto*
- *random_big_int (seed:int4& -const) : auto*
- *random_uint (seed:int4& -const) : auto*
- *random_int4 (seed:int4& -const) : auto*
- *random_float (seed:int4& -const) : auto*
- *random_float4 (seed:int4& -const) : auto*

random_seed(*seed: int const*)

random_seed returns auto

argument	argument type
seed	int const

constructs seed vector out of single integer seed

random_seed2D (*seed: int4&; co: int2 const; cf: int const*)

random_seed2D returns auto

argument	argument type
seed	int4&
co	int2 const
cf	int const

constructs seed vector out of 2d screen coordinates and frame counter *cf*

random_int (*seed: int4&*)

random_int returns auto

argument	argument type
seed	int4&

random integer 0..32767 (LCG_RAND_MAX)

random_big_int (*seed: int4&*)

random_big_int returns auto

argument	argument type
seed	int4&

random integer 0..32768*32768-1 (LCG_RAND_MAX_BIG)

random_uint (*seed: int4&*)

random_uint returns auto

argument	argument type
seed	int4&

random integer 0..32768*32768-1 (LCG_RAND_MAX_BIG)

random_int4 (*seed: int4&*)

random_int4 returns auto

argument	argument type
seed	int4&

random int4, each component is 0..32767 (LCG_RAND_MAX)

random_float (*seed: int4&*)

random_float returns auto

argument	argument type
seed	int4&

random float 0..1

random_float4 (*seed: int4&*)

random_float4 returns auto

argument	argument type
seed	int4&

random float4, each component is 0..1

7.3 Random iterators

- *each_random_uint (rnd_seed:int const) : iterator<uint>*

each_random_uint (*rnd_seed: int const*)

each_random_uint returns iterator<uint>

argument	argument type
rnd_seed	int const

endless iterator of random uints

7.4 Specific distributions

- *random_unit_vector (seed:int4& -const) : auto*
- *random_in_unit_sphere (seed:int4& -const) : auto*
- *random_in_unit_disk (seed:int4& -const) : auto*

random_unit_vector (*seed: int4&*)

random_unit_vector returns auto

argument	argument type
seed	int4&

random float3 unit vector (length=1.)

random_in_unit_sphere (*seed: int4&*)

random_in_unit_sphere returns auto

argument	argument type
seed	int4&

random float3 unit vector (length=1) which happens to be inside a sphere R=1

random_in_unit_disk (*seed: int4&*)

random_in_unit_disk returns auto

argument	argument type
seed	int4&

random float3 unit vector (length=1) which happens to be inside a disk R=1, Z=0

CHAPTER

EIGHT

NETWORK SOCKET LIBRARY

The NETWORK module implements basic TCP socket listening server (currently only one connection). It would eventually be expanded to support client as well.

It its present form its used in Daslang Visual Studio Code plugin and upcoming debug server.

All functions and symbols are in "network" module, use require to get access to it.

```
require network
```

8.1 Handled structures

NetworkServer

Base impliemntation of the server.

8.2 Classes

Server

Single socket listener combined with single socket connection.

it defines as follows

_server : smart_ptr< *network::NetworkServer* >

Server.**make_server_adapter** (*self: Server*)

Creates new instance of the server adapter. Adapter is responsible for communicating with the Server class.

Server.**init** (*self: Server; port: int const*)

init returns bool

argument	argument type
self	*network::Server*
port	int const

Initializes server with specific port

Server.**restore** (*self: Server; shared_orphan: smart_ptr<network::NetworkServer>&*)

argument	argument type
self	*network::Server*
shared_orphan	smart_ptr< *network::NetworkServer* >&

Restore server state from after the context switch.

`Server.`**`save`**(*self: Server; shared_orphan: smart_ptr<network::NetworkServer>&*)

argument	argument type
self	*network::Server*
shared_orphan	smart_ptr< *network::NetworkServer* >&

Saves server to orphaned state to support context switching and live reloading. The idea is that server is saved to the orphaned state, which is not part of the context state.

`Server.`**`has_session`**(*self: Server*)

has_session returns bool

Returns true if network session already exists. This is used to determine if the server should be initialized or not.

`Server.`**`is_open`**(*self: Server*)

is_open returns bool

Returns true if server is listening to the port.

`Server.`**`is_connected`**(*self: Server*)

is_connected returns bool

Returns true if server is connected to the client.

`Server.`**`tick`**(*self: Server*)

This needs to be called periodically to support the server communication and connections.

`Server.`**`send`**(*self: Server; data: uint8? const; size: int const*)

send returns bool

argument	argument type
self	*network::Server*
data	uint8? const
size	int const

Send data.

`Server.`**`onConnect`**(*self: Server*)

This callback is called when server accepts the connection.

`Server`.**onDisconnect**(*self: Server*)

This callback is called when server or client drops the connection.

`Server`.**onData**(*self: Server; buf: uint8? const; size: int const*)

argument	argument type
self	*network::Server*
buf	uint8? const
size	int const

This callback is called when data is received from the client.

`Server`.**onError**(*self: Server; msg: string const; code: int const*)

argument	argument type
self	*network::Server*
msg	string const
code	int const

This callback is called on any error.

`Server`.**onLog**(*self: Server; msg: string const*)

argument	argument type
self	*network::Server*
msg	string const

This is how server logs are printed.

8.3 Low lever NetworkServer IO

- *make_server (class:void? const implicit;info:rtti::StructInfo const? const implicit;context:__context const) : bool*
- *server_init (server:smart_ptr<network::NetworkServer> const implicit;port:int const;context:__context const;at:__lineInfo const) : bool*
- *server_is_open (server:smart_ptr<network::NetworkServer> const implicit;context:__context const;at:__lineInfo const) : bool*

- *server_is_connected (server:smart_ptr<network::NetworkServer> const implicit;context:__context const;at:__lineInfo const) : bool*
- *server_tick (server:smart_ptr<network::NetworkServer> const implicit;context:__context const;at:__lineInfo const) : void*
- *server_send (server:smart_ptr<network::NetworkServer> const implicit;data:uint8? const implicit;size:int const;context:__context const;at:__lineInfo const) : bool*
- *server_restore (server:smart_ptr<network::NetworkServer> const implicit;class:void? const implicit;info:rtti::StructInfo const? const implicit;context:__context const;at:__lineInfo const) : void*

make_server (*class: void? const implicit; info: rtti::StructInfo const? const implicit*)

make_server returns bool

argument	argument type
class	void? const implicit
info	*rtti::StructInfo* const? const implicit

Creates new instance of the server.

server_init (*server: smart_ptr<network::NetworkServer> const implicit; port: int const*)

server_init returns bool

argument	argument type
server	smart_ptr< *network::NetworkServer* > const implicit
port	int const

Initializes server with given port.

server_is_open (*server: smart_ptr<network::NetworkServer> const implicit*)

server_is_open returns bool

argument	argument type
server	smart_ptr< *network::NetworkServer* > const implicit

Returns true if server is listening to the port.

server_is_connected (*server: smart_ptr<network::NetworkServer> const implicit*)

server_is_connected returns bool

argument	argument type
server	smart_ptr< *network::NetworkServer* > const implicit

Returns true if server is connected to the client.

server_tick (*server: smart_ptr<network::NetworkServer> const implicit*)

argument	argument type
server	smart_ptr< *network::NetworkServer* > const implicit

This needs to be called periodically for the server to work.

server_send (*server: smart_ptr<network::NetworkServer> const implicit; data: uint8? const implicit; size: int const*)

server_send returns bool

argument	argument type
server	smart_ptr< *network::NetworkServer* > const implicit
data	uint8? const implicit
size	int const

Sends data from server to the client.

server_restore (*server: smart_ptr<network::NetworkServer> const implicit; class: void? const implicit; info: rtti::StructInfo const? const implicit*)

argument	argument type
server	smart_ptr< *network::NetworkServer* > const implicit
class	void? const implicit
info	*rtti::StructInfo* const? const implicit

Restores server from orphaned state.

CHAPTER

NINE

URI MANIPULATION LIBRARY BASED ON URIPARSER

The URIPARSER module exposes uriParser library https://uriparser.github.io to Daslang.

All functions and symbols are in “uriparser” module, use require to get access to it.

```
require uriparser
```

9.1 Handled structures

UriTextRangeA

Range of text in the URI.

UriIp4Struct

UriIp4Struct fields are

data	uint8[4]

IPv4 address portion of the URI.

UriIp6Struct

UriIp6Struct fields are

data	uint8[16]

IPv6 address porition of the URI.

UriHostDataA

UriHostDataA fields are

ipFuture	*uriparser::UriTextRangeA*
ip4	*uriparser::UriIp4Struct* ?
ip6	*uriparser::UriIp6Struct* ?

Host data portion of the URI (IPv4 or IPv6, or some future data).

UriPathSegmentStructA

UriPathSegmentStructA fields are

next	*uriparser::UriPathSegmentStructA* ?
text	*uriparser::UriTextRangeA*

Part of the path portion of the URI.

UriUriA

UriUriA fields are

query	*uriparser::UriTextRangeA*
absolutePath	int
fragment	*uriparser::UriTextRangeA*
userInfo	*uriparser::UriTextRangeA*
hostText	*uriparser::UriTextRangeA*
scheme	*uriparser::UriTextRangeA*
hostData	*uriparser::UriHostDataA*
portText	*uriparser::UriTextRangeA*
pathTail	*uriparser::UriPathSegmentStructA* ?
owner	int
pathHead	*uriparser::UriPathSegmentStructA* ?

URI base class, contains all URI data.

Uri

Uri fields are

uri	*uriparser::UriUriA*

Uri property operators are

empty	bool
size	int
status	int

URI implementation.

9.2 Initialization and finalization

- *Uri () : uriparser::Uri*
- *using (arg0:block<(var arg0:uriparser::Uri# explicit):void> const implicit) : void*
- *Uri (arg0:string const implicit) : uriparser::Uri*
- *using (arg0:string const implicit;arg1:block<(var arg0:uriparser::Uri# explicit):void> const implicit) : void*
- *finalize (uri:uriparser::Uri implicit) : void*
- *clone (dest:uriparser::Uri implicit;src:uriparser::Uri const implicit) : void*

Uri ()

Uri returns *uriparser::Uri*

Creates new URI.

using (*arg0: block<(var arg0:uriparser::Uri# explicit):void> const implicit*)

argument	argument type
arg0	block<(*uriparser::Uri* #):void> const implicit

Creates scoped URI variable.

Uri (*arg0: string const implicit*)

Uri returns *uriparser::Uri*

argument	argument type
arg0	string const implicit

Creates new URI.

using (*arg0: string const implicit; arg1: block<(var arg0:uriparser::Uri# explicit):void> const implicit*)

argument	argument type
arg0	string const implicit
arg1	block<(*uriparser::Uri* #):void> const implicit

Creates scoped URI variable.

finalize (*uri: Uri implicit*)

argument	argument type
uri	*uriparser::Uri* implicit

Finalizer for the URI.

clone (*dest: Uri implicit; src: Uri const implicit*)

argument	argument type
dest	*uriparser::Uri* implicit
src	*uriparser::Uri* const implicit

Clones the URI.

9.3 Escape and unescape

- *escape_uri (uriStr:string const implicit;spaceToPlus:bool const;normalizeBreaks:bool const;context:__context const) : string*
- *unescape_uri (uriStr:string const implicit;context:__context const) : string*

escape_uri (*uriStr: string const implicit; spaceToPlus: bool const; normalizeBreaks: bool const*)

escape_uri returns string

argument	argument type
uriStr	string const implicit
spaceToPlus	bool const
normalizeBreaks	bool const

Adds escape characters to the URI.

unescape_uri (*uriStr: string const implicit*)

unescape_uri returns string

argument	argument type
uriStr	string const implicit

Remove escape characters from the URI.

9.4 Uri manipulations

- *strip_uri (uri:uriparser::Uri const implicit;query:bool const;fragment:bool const) : uriparser::Uri*
- *add_base_uri (base:uriparser::Uri const implicit;relative:uriparser::Uri const implicit) : uriparser::Uri*
- *remove_base_uri (base:uriparser::Uri const implicit;relative:uriparser::Uri const implicit) : uriparser::Uri*
- *normalize (uri:uriparser::Uri implicit) : bool*
- *string (uri:uriparser::Uri const implicit;context:__context const) : string*
- *string (range:uriparser::UriTextRangeA const implicit;context:__context const) : string*
- *uri_for_each_query_kv (uri:uriparser::Uri const implicit;block:block<(var arg0:string#;var arg1:string#):void> const implicit;context:__context const;lineinfo:__lineInfo const) : void*
- *normalize_uri (uriStr:string const implicit;context:__context const) : string*

strip_uri (*uri: Uri const implicit; query: bool const; fragment: bool const*)

strip_uri returns *uriparser::Uri*

argument	argument type
uri	*uriparser::Uri* const implicit
query	bool const
fragment	bool const

Removes query and fragment from the URI.

add_base_uri (*base: Uri const implicit; relative: Uri const implicit*)

add_base_uri returns *uriparser::Uri*

argument	argument type
base	*uriparser::Uri* const implicit
relative	*uriparser::Uri* const implicit

Adds *base* URI to the *relative* URI.

remove_base_uri (*base: Uri const implicit; relative: Uri const implicit*)

remove_base_uri returns *uriparser::Uri*

argument	argument type
base	*uriparser::Uri* const implicit
relative	*uriparser::Uri* const implicit

Removes *base* URI from the *relative* URI.

normalize (*uri: Uri implicit*)

normalize returns bool

argument	argument type
uri	*uriparser::Uri* implicit

Normalizes URI, i.e. removes redundant / and . characters.

string (*uri: Uri const implicit*)

string returns string

argument	argument type
uri	*uriparser::Uri* const implicit

Converts URI to string.

string (*range: UriTextRangeA const implicit*)

string returns string

argument	argument type
range	*uriparser::UriTextRangeA* const implicit

Converts URI to string.

uri_for_each_query_kv (*uri: Uri const implicit; block: block<(var arg0:string#;var arg1:string#):void> const implicit*)

argument	argument type
uri	*uriparser::Uri* const implicit
block	block<(string#;string#):void> const implicit

Iterates over the URI query parameters.

normalize_uri (*uriStr: string const implicit*)

normalize_uri returns string

argument	argument type
uriStr	string const implicit

Normalizes URI. i.e. removes redundant / and . characters.

9.5 File name conversions

- *to_unix_file_name (uri:uriparser::Uri const implicit;context:__context const) : string*
- *to_windows_file_name (uri:uriparser::Uri const implicit;context:__context const) : string*
- *to_file_name (uri:uriparser::Uri const implicit;context:__context const) : string*
- *uri_from_file_name (filename:string const implicit) : uriparser::Uri*
- *uri_from_windows_file_name (filename:string const implicit) : uriparser::Uri*
- *uri_from_unix_file_name (filename:string const implicit) : uriparser::Uri*
- *uri_to_unix_file_name (uriStr:string const implicit;context:__context const) : string*
- *uri_to_windows_file_name (uriStr:string const implicit;context:__context const) : string*
- *unix_file_name_to_uri (uriStr:string const implicit;context:__context const) : string*
- *windows_file_name_to_uri (uriStr:string const implicit;context:__context const) : string*
- *uri_to_file_name (uriStr:string const implicit;context:__context const) : string*
- *file_name_to_uri (uriStr:string const implicit;context:__context const) : string*

to_unix_file_name (*uri: Uri const implicit*)

to_unix_file_name returns string

argument	argument type
uri	*uriparser::Uri* const implicit

Converts URI to Unix file name.

to_windows_file_name (*uri: Uri const implicit*)

to_windows_file_name returns string

argument	argument type
uri	*uriparser::Uri* const implicit

Converts URI to Windows file name.

to_file_name (*uri: Uri const implicit*)

to_file_name returns string

argument	argument type
uri	*uriparser::Uri* const implicit

Converts URI to the current platform file name.

uri_from_file_name (*filename: string const implicit*) uri_from_file_name returns *uriparser::Uri*

argument	argument type
filename	string const implicit

Converts current platform file name to URI.

uri_from_windows_file_name (*filename: string const implicit*)

uri_from_windows_file_name returns *uriparser::Uri*

argument	argument type
filename	string const implicit

Converts Windows file name to URI.

uri_from_unix_file_name (*filename: string const implicit*)

uri_from_unix_file_name returns *uriparser::Uri*

argument	argument type
filename	string const implicit

Converts Unix file name to URI.

uri_to_unix_file_name (*uriStr: string const implicit*)

uri_to_unix_file_name returns string

argument	argument type
uriStr	string const implicit

Converts URI to Unix file name.

uri_to_windows_file_name (*uriStr: string const implicit*)

uri_to_windows_file_name returns string

argument	argument type
uriStr	string const implicit

Converts URI to Windows file name.

unix_file_name_to_uri (*uriStr: string const implicit*) unix_file_name_to_uri returns string

argument	argument type
uriStr	string const implicit

Converts Unix file name to URI.

windows_file_name_to_uri (*uriStr: string const implicit*)

windows_file_name_to_uri returns string

argument	argument type
uriStr	string const implicit

Converts Windows file name to URI.

uri_to_file_name (*uriStr: string const implicit*)

uri_to_file_name returns string

argument	argument type
uriStr	string const implicit

Converts URI to the current platform file name.

file_name_to_uri (*uriStr: string const implicit*)

file_name_to_uri returns string

argument	argument type
uriStr	string const implicit

Converts current file name to URI.

9.6 GUID

- *make_new_guid (context:__context const;at:__lineInfo const) : string*

make_new_guid ()

make_new_guid returns string

Generates new GUID.

CHAPTER

TEN

BOOST PACKAGE FOR THE URI PARSER

The uriparser_boost module implements additional infrastructure for the URI parser.

All functions and symbols are in “uriparser_boost” module, use require to get access to it.

```
require daslib/uriparser_boost
```

10.1 Split and compose

- *uri_split_full_path (uri:uriparser::Uri const implicit) : array<string>*
- *uri_compose_query (query:table<string;string> const) : string*
- *uri_compose_query_in_order (query:table<string;string> const) : string const*
- *uri_compose (scheme:string const;userInfo:string const;hostText:string const;portText:string const;path:string const;query:string const;fragment:string const) : uriparser::Uri*

uri_split_full_path (*uri: Uri const implicit*)

uri_split_full_path returns array<string>

argument	argument type
uri	*uriparser::Uri* const implicit

Split the full path of a URI into its components.

uri_compose_query (*query: table<string;string> const*)

uri_compose_query returns string

argument	argument type
query	table<string;string> const

Compose a query string from a table of key-value pairs.

uri_compose_query_in_order (*query: table<string;string> const*) uri_compose_query_in_order returns string const

argument	argument type
query	table<string;string> const

Compose a query string from a table of key-value pairs, in the sorted order.

uri_compose (*scheme: string const; userInfo: string const; hostText: string const; portText: string const; path: string const; query: string const; fragment: string const*)

uri_compose returns *uriparser::Uri*

argument	argument type
scheme	string const
userInfo	string const
hostText	string const
portText	string const
path	string const
query	string const
fragment	string const

Compose a URI from its components.

10.2 Component accessors

- *scheme (uri:uriparser::Uri const implicit) : string*
- *user_info (uri:uriparser::Uri const implicit) : string*
- *host (uri:uriparser::Uri const implicit) : string*
- *port (uri:uriparser::Uri const implicit) : string*
- *path (uri:uriparser::Uri const implicit) : string*
- *query (uri:uriparser::Uri const implicit) : string*
- *fragment (uri:uriparser::Uri const implicit) : string*

scheme (*uri: Uri const implicit*)

scheme returns string

argument	argument type
uri	*uriparser::Uri* const implicit

Returns the scheme of a URI.

user_info (*uri: Uri const implicit*)

user_info returns string

argument	argument type
uri	*uriparser::Uri* const implicit

Return the user info of a URI.

host (*uri: Uri const implicit*)

host returns string

argument	argument type
uri	*uriparser::Uri* const implicit

Return the host of a URI.

port (*uri: Uri const implicit*)

port returns string

argument	argument type
uri	*uriparser::Uri* const implicit

Return the port of a URI.

path (*uri: Uri const implicit*)

path returns string

argument	argument type
uri	*uriparser::Uri* const implicit

Return the path of a URI.

query (*uri: Uri const implicit*)

query returns string

argument	argument type
uri	*uriparser::Uri* const implicit

Return the query of a URI.

fragment (*uri: Uri const implicit*)

fragment returns string

argument	argument type
uri	*uriparser::Uri* const implicit

Return the fragment of a URI.

CHAPTER

ELEVEN

RUNTIME TYPE INFORMATION LIBRARY

The RTTI module reflects runtime type information to Daslang. It also exposes Daslang compiler infrastructure to Daslang runtime.

All functions and symbols are in "rtti" module, use require to get access to it.

```
require rtti
```

11.1 Type aliases

ProgramFlags is a bitfield

field	bit	value
failToCompile	0	1
_unsafe	1	2
isCompiling	2	4
isSimulating	3	8
isCompilingMacros	4	16
needMacroModule	5	32

Flags which represent state of the *Program* object, both during and after compilation.

context_category_flags is a bitfield

field	bit	value
dead	0	1
debug_context	1	2
thread_clone	2	4
job_clone	3	8
opengl	4	16
debugger_tick	5	32
debugger_attached	6	64
macro_context	7	128
folding_context	8	256
audio	9	512

Flags which specify type of the *Context*.

`TypeInfoFlags is a bitfield`

field	bit	value
ref	0	1
refType	1	2
canCopy	2	4
isPod	3	8
isRawPod	4	16
isConst	5	32
isTemp	6	64
isImplicit	7	128
refValue	8	256
hasInitValue	9	512
isSmartPtr	10	1024
isSmartPtrNative	11	2048

continues on next page

Table 1 – continued from previous page

field	bit	value
isHandled	12	4096
heapGC	13	8192
stringHeapGC	14	16384
lockCheck	15	32768

Flags which specify properties of the *TypeInfo* object (any rtti type).

StructInfoFlags is a bitfield

field	bit	value
_class	0	1
_lambda	1	2
heapGC	2	4
stringHeapGC	3	8
lockCheck	4	16

Flags which represent properties of the *StructInfo* object (rtti object which represents structure type).

ModuleFlags is a bitfield

field	bit	value
builtIn	0	1
promoted	1	2
isPublic	2	4
isModule	3	8
isSolidContext	4	16
doNotAllowUnsafe	5	32

Flags which represent the module's state.

AnnotationDeclarationFlags is a bitfield

field	bit	value
inherited	0	1

Flags which represent properties of the *AnnotationDeclaration* object.

RttiValue is a variant type

tBool	bool
tInt	int
tUInt	uint
tInt64	int64
tUInt64	uint64
tFloat	float
tDouble	double
tString	string
nothing	any

Variant type which represents value of any annotation arguments and variable annotations.

FileAccessPtr = smart_ptr<rtti::FileAccess>

smart_ptr<FileAccess>, i.e pointer to the *FileAccess* object.

11.2 Constants

FUNCINFO_INIT = 0x1

Function flag which indicates that function is called during the *Context* initialization.

FUNCINFO_BUILTIN = 0x2

Function flag which indicates that function is a built-in function.

FUNCINFO_PRIVATE = 0x4

Function flag which indicates that function is private.

FUNCINFO_SHUTDOWN = 0x8

Function flag which indicates that function is called during the *Context* shutdown.

FUNCINFO_LATE_INIT = 0x20

Function flag which indicates that function initialization is ordered via custom init order.

11.3 Enumerations

CompilationError

unspecified	0
mismatching_parentheses	10001
mismatching_curly_bracers	10002
string_constant_exceeds_file	10003
string_constant_exceeds_line	10004
unexpected_close_comment	10005
integer_constant_out_of_range	10006
comment_contains_eof	10007
invalid_escape_sequence	10008
invalid_line_directive	10009
syntax_error	20000
malformed_ast	20001
invalid_type	30101
invalid_return_type	30102
invalid_argument_type	30103
invalid_structure_field_type	30104
invalid_array_type	30105
invalid_table_type	30106
invalid_argument_count	30107
invalid_variable_type	30108
invalid_new_type	30109
invalid_index_type	30110
invalid_annotation	30111
invalid_swizzle_mask	30112

continues on next page

Table 2 – continued from previous page

invalid_initialization_type	30113
invalid_with_type	30114
invalid_override	30115
invalid_name	30116
invalid_array_dimension	30117
invalid_iteration_source	30118
invalid_loop	30119
invalid_label	30120
invalid_enumeration	30121
invalid_option	30122
invalid_member_function	30123
function_already_declared	30201
argument_already_declared	30202
local_variable_already_declared	30203
global_variable_already_declared	30204
structure_field_already_declared	30205
structure_already_declared	30206
structure_already_has_initializer	30207
enumeration_already_declared	30208
enumeration_value_already_declared	30209
type_alias_already_declared	30210
field_already_initialized	30211
type_not_found	30301
structure_not_found	30302
operator_not_found	30303

continues on next page

Table 2 – continued from previous page

function_not_found	30304
variable_not_found	30305
handle_not_found	30306
annotation_not_found	30307
enumeration_not_found	30308
enumeration_value_not_found	30309
type_alias_not_found	30310
bitfield_not_found	30311
cant_initialize	30401
cant_dereference	30501
cant_index	30502
cant_get_field	30503
cant_write_to_const	30504
cant_move_to_const	30505
cant_write_to_non_reference	30506
cant_copy	30507
cant_move	30508
cant_pass_temporary	30509
condition_must_be_bool	30601
condition_must_be_static	30602
cant_pipe	30701
invalid_block	30801
return_or_break_in_finally	30802
module_not_found	30901
module_already_has_a_name	30902
cant_new_handle	31001

continues on next page

Table 2 – continued from previous page

bad_delete	31002
cant_infer_generic	31100
cant_infer_missing_initializer	31101
cant_infer_mismatching_restrictions	31102
invalid_cast	31200
incompatible_cast	31201
unsafe	31300
index_out_of_range	31400
expecting_return_value	32101
not_expecting_return_value	32102
invalid_return_semantics	32103
invalid_yield	32104
typeinfo_reference	39901
typeinfo_auto	39902
typeinfo_undefined	39903
typeinfo_dim	39904
typeinfo_macro_error	39905
static_assert_failed	40100
run_failed	40101
annotation_failed	40102
concept_failed	40103
not_all_paths_return_value	40200
assert_with_side_effects	40201
only_fast_aot_no_cpp_name	40202
aot_side_effects	40203

continues on next page

Table 2 – continued from previous page

no_global_heap	40204
no_global_variables	40205
unused_function_argument	40206
unsafe_function	40207
too_many_infer_passes	41000
missing_node	50100

Enumeration which represents error type for each of the errors which compiler returns and various stages.

Type

none	0
autoinfer	1
alias	2
option	3
fakeContext	4
fakeLineInfo	5
anyArgument	6
tVoid	7
tBool	8
tInt64	13
tUInt64	14
tInt	15
tInt2	16
tInt3	17
tInt4	18
tUInt	19
tUInt2	20
tUInt3	21

continues on next page

Table 3 – continued from previous page

tUInt4	22
tFloat	23
tFloat2	24
tFloat3	25
tFloat4	26
tDouble	27
tRange	28
tURange	29
tRange64	30
tURange64	31
tString	32
tStructure	33
tHandle	34
tEnumeration	35
tPointer	39
tFunction	40
tLambda	41
tIterator	42
tArray	43
tTable	44
tBlock	45
tInt8	9
tUInt8	10
tInt16	11
tUInt16	12

continues on next page

Table 3 – continued from previous page

tTuple	46
tEnumeration8	36
tEnumeration16	37
tVariant	47
tBitfield	38

One of the fundamental (base) types of any type object.

RefMatters

no	0
yes	1

Yes or no flag which indicates if reference flag of the type matters (during comparison).

ConstMatters

no	0
yes	1

Yes or no flag which indicates if constant flag of the type matters (during comparison).

TemporaryMatters

no	0
yes	1

Yes or no flag which indicates if temporary flag of the type matters (during comparison).

11.4 Handled structures

FileInfo

FileInfo fields are

name	*builtin::das_string*
tabSize	int

Information about a single file stored in the *FileAccess* object.

LineInfo

LineInfo fields are

last_column	uint
line	uint
last_line	uint
column	uint
fileInfo	*rtti::FileInfo* ?

Information about a section of the file stored in the *FileAccess* object.

Context

Context fields are

breakOnException	bool
exception	string const
category	*context_category_flags*
alwaysStackWalkOnException	bool
last_exception	string const
contextMutex	*rtti::recursive_mutex* ?
name	*builtin::das_string*
exceptionAt	*rtti::LineInfo*

Context property operators are

totalFunctions	int
totalVariables	int
getCodeAllocatorId	uint64

Object which holds single Daslang Context. Context is the result of the simulation of the Daslang program.

Error

Error fields are

fixme	*builtin::das_string*
at	*rtti::LineInfo*
what	*builtin::das_string*
extra	*builtin::das_string*
cerr	*rtti::CompilationError*

Object which holds information about compilation error or exception.

FileAccess

Object which holds collection of files as well as means to access them (Project).

Module

Module fields are

moduleFlags	*ModuleFlags*
name	*builtin::das_string*

Collection of types, aliases, functions, classes, macros etc under a single namespace.

ModuleGroup

Collection of modules.

AnnotationArgument

AnnotationArgument fields are

fValue	float
at	*rtti::LineInfo*
iValue	int
name	*builtin::das_string*
sValue	*builtin::das_string*
bValue	bool
basicType	*rtti::Type*

Single argument of the annotation, typically part of the *AnnotationArgumentList*.

Program

Program fields are

thisModuleName	*builtin::das_string*
_options	*rtti::AnnotationArgumentList*
errors	vector<Error>
flags	*ProgramFlags*

Object representing full information about Daslang program during and after compilation (but not the simulated result of the program).

Annotation

Annotation fields are

_module	*rtti::Module* ?
cppName	*builtin::das_string*
name	*builtin::das_string*

Annotation property operators are

isTypeAnnotation	bool
isBasicStructureAnnotation	bool
isStructureAnnotation	bool
isStructureTypeAnnotation	bool
isFunctionAnnotation	bool
isEnumerationAnnotation	bool

Handled type or macro.

AnnotationDeclaration

AnnotationDeclaration fields are

annotation	smart_ptr< *rtti::Annotation* >
arguments	*rtti::AnnotationArgumentList*
at	*rtti::LineInfo*
flags	*AnnotationDeclarationFlags*

Annotation declaration, its location, and arguments.

TypeAnnotation

TypeAnnotation fields are

_module	*rtti::Module* ?
cppName	*builtin::das_string*
name	*builtin::das_string*

TypeAnnotation property operators are

is_any_vector	bool
canMove	bool
canCopy	bool
canClone	bool
isPod	bool
isRawPod	bool
isRefType	bool
hasNonTrivialCtor	bool
hasNonTrivialDtor	bool
hasNonTrivialCopy	bool
canBePlacedInContainer	bool
isLocal	bool
canNew	bool
canDelete	bool
needDelete	bool
canDeletePtr	bool
isIterable	bool
isShareable	bool
isSmart	bool

continues on next page

Table 4 – continued from previous page

avoidNullPtr	bool
sizeOf	uint64
alignOf	uint64

Handled type.

BasicStructureAnnotation

BasicStructureAnnotation fields are

name	*builtin::das_string*
cppName	*builtin::das_string*

BasicStructureAnnotation property operators are

fieldCount	int

Handled type which represents structure-like object.

EnumValueInfo

EnumValueInfo fields are

value	int64
name	string const

Single element of enumeration, its name and value.

EnumInfo

EnumInfo fields are

count	uint
name	string const
module_name	string const
hash	uint64

Type object which represents enumeration.

StructInfo

StructInfo fields are

init_mnh	uint64
size	uint
count	uint
name	string const
module_name	string const
hash	uint64
flags	*StructInfoFlags*

Type object which represents structure or class.

TypeInfo

TypeInfo fields are

argTypes	*rtti::TypeInfo* ??
size	uint
secondType	*rtti::TypeInfo* ?
dimSize	uint
hash	uint64
argNames	string const?
argCount	uint
basicType	*rtti::Type*
firstType	*rtti::TypeInfo* ?
flags	*TypeInfoFlags*

TypeInfo property operators are

enumType	*rtti::EnumInfo* ?
isRef	bool
isRefType	bool
isRefValue	bool
canCopy	bool
isPod	bool
isRawPod	bool
isConst	bool
isTemp	bool
isImplicit	bool
annotation	*rtti::TypeAnnotation* ?
structType	*rtti::StructInfo* ?

Object which represents any Daslang type.

VarInfo

VarInfo fields are

argTypes	*rtti::TypeInfo* ??
size	uint
value	any
secondType	*rtti::TypeInfo* ?
dimSize	uint
name	string const
hash	uint64
argNames	string const?
argCount	uint
sValue	string
offset	uint
basicType	*rtti::Type*
annotation_arguments	*rtti::AnnotationArguments* const? const
firstType	*rtti::TypeInfo* ?
flags	*TypeInfoFlags*

Object which represents variable declaration.

LocalVariableInfo

LocalVariableInfo fields are

visibility	*rtti::LineInfo*
argTypes	*rtti::TypeInfo* ??
size	uint
secondType	*rtti::TypeInfo* ?
dimSize	uint
localFlags	LocalVariableInfoFlags
stackTop	uint
name	string const
hash	uint64
argNames	string const?
argCount	uint
basicType	*rtti::Type*
firstType	*rtti::TypeInfo* ?
flags	*TypeInfoFlags*

Object which represents local variable declaration.

FuncInfo

FuncInfo fields are

locals	*rtti::LocalVariableInfo* ??
stackSize	uint
result	*rtti::TypeInfo* ?
count	uint
globals	*rtti::VarInfo* ??
cppName	string const
name	string const
globalCount	uint
hash	uint64
localCount	uint
flags	uint

Object which represents function declaration.

SimFunction

SimFunction fields are

stackSize	uint
mangledNameHash	uint64
mangledName	string
name	string
debugInfo	*rtti::FuncInfo* ?
flags	SimFunctionFlags

SimFunction property operators are

lineInfo	*rtti::LineInfo* const? const

Object which represents simulated function in the *Context*.

CodeOfPolicies

CodeOfPolicies fields are

aot_module	bool
fail_on_no_aot	bool
jit	bool
fail_on_lack_of_aot_export	bool
profiler	bool
debugger	bool
aot_order_side_effects	bool
threadlock_context	bool
macro_context_collect	bool
rtti	bool
ignore_shared_modules	bool
no_deprecated	bool
aot	bool
allow_shared_lambda	bool
allow_local_variable_shadowing	bool
multiple_contexts	bool
heap_size_hint	uint
profile_module	*builtin::das_string*
no_init	bool
always_report_candidates_threshold	int
persistent_heap	bool
no_global_heap	bool
intern_strings	bool
no_optimizations	bool
allow_block_variable_shadowing	bool
no_unused_function_arguments	bool

continues on next page

Table 5 – continued from previous page

stack	uint
smart_pointer_by_value_unsafe	bool
no_unused_block_arguments	bool
export_all	bool
solid_context	bool
no_global_variables	bool
completion	bool
string_heap_size_hint	uint
macro_context_persistent_heap	bool
no_unsafe	bool
local_ref_is_unsafe	bool
no_aliasing	bool
no_global_variables_at_all	bool
strict_smart_pointers	bool
only_fast_aot	bool
debug_module	*builtin::das_string*
strict_unsafe_delete	bool
default_module_public	bool

Object which holds compilation and simulation settings and restrictions.

11.5 Typeinfo macros

rtti_typeinfo

Generates *TypeInfo* for the given expression or type.

11.6 Handled types

recursive_mutex

Holds system-specific recursive mutex object (typically std::recursive_mutex).

AnnotationArguments

List of annotation arguments.

AnnotationArgumentList

List of annotation arguments and properties.

AnnotationList

List of all annotations attached to the object (function or structure).

11.7 Initialization and finalization

- *LineInfo () : rtti::LineInfo*
- *LineInfo (arg0:rtti::FileInfo? const implicit;arg1:int const;arg2:int const;arg3:int const;arg4:int const) : rtti::LineInfo*
- *using (arg0:block<(var arg0:rtti::recursive_mutex explicit):void> const implicit) : void*
- *CodeOfPolicies () : rtti::CodeOfPolicies*
- *using (arg0:block<(var arg0:rtti::CodeOfPolicies explicit):void> const implicit) : void*
- *using (arg0:block<(var arg0:rtti::ModuleGroup explicit):void> const implicit) : void*
- *RttiValue_nothing () : auto*

LineInfo()

LineInfo returns *rtti::LineInfo*

LineInfo initializer.

LineInfo(*arg0: rtti::FileInfo? const implicit; arg1: int const; arg2: int const; arg3: int const; arg4: int const*)

LineInfo returns *rtti::LineInfo*

argument	argument type
arg0	*rtti::FileInfo* ? const implicit
arg1	int const
arg2	int const
arg3	int const
arg4	int const

LineInfo initializer.

using (*arg0: block<(var arg0:rtti::recursive_mutex explicit):void> const implicit*)

argument	argument type
arg0	block<(*rtti::recursive_mutex*):void> const implicit

Creates object which can be used inside of the block scope.

CodeOfPolicies ()

CodeOfPolicies returns *rtti::CodeOfPolicies*

CodeOfPolicies initializer.

using (*arg0: block<(var arg0:rtti::CodeOfPolicies explicit):void> const implicit*)

argument	argument type
arg0	block<(*rtti::CodeOfPolicies*):void> const implicit

Creates object which can be used inside of the block scope.

using (*arg0: block<(var arg0:rtti::ModuleGroup explicit):void> const implicit*)

argument	argument type
arg0	block<(*rtti::ModuleGroup*):void> const implicit

Creates object which can be used inside of the block scope.

RttiValue_nothing ()

RttiValue_nothing returns auto

Constructs new RttiValue of type ‘nothing’.

11.8 Type access

- *get_dim (typeinfo:rtti::TypeInfo const implicit;index:int const;context:__context const;at:__lineInfo const) : int*
- *get_dim (typeinfo:rtti::VarInfo const implicit;index:int const;context:__context const;at:__lineInfo const) : int*
- *builtin_is_same_type (a:rtti::TypeInfo const? const implicit;b:rtti::TypeInfo const? const implicit;refMatters:rtti::RefMatters const;cosntMatters:rtti::ConstMatters const;tempMatters:rtti::TemporaryMatters const;topLevel:bool const) : bool*
- *get_type_size (type:rtti::TypeInfo? const implicit) : int*
- *get_type_align (type:rtti::TypeInfo? const implicit) : int*
- *is_compatible_cast (from:rtti::StructInfo const? const implicit;to:rtti::StructInfo const? const implicit) : bool*
- *get_das_type_name (type:rtti::Type const;context:__context const) : string*
- *is_same_type (a:rtti::TypeInfo const;b:rtti::TypeInfo const;refMatters:rtti::RefMatters const;constMatters:rtti::ConstMatters const;temporaryMatters:rtti::TemporaryMatters const;topLevel:bool const) : auto*
- *is_compatible_cast (a:rtti::StructInfo const;b:rtti::StructInfo const) : auto*
- *each_dim (info:rtti::TypeInfo const) : auto*
- *each_dim (info:rtti::VarInfo const) : auto*
- *arg_types (info:rtti::TypeInfo const) : auto*
- *arg_types (info:rtti::VarInfo const) : auto*
- *arg_names (info:rtti::TypeInfo const) : auto*
- *arg_names (info:rtti::VarInfo const) : auto*

get_dim(*typeinfo: TypeInfo const implicit; index: int const*)

get_dim returns int

argument	argument type
typeinfo	*rtti::TypeInfo* const implicit
index	int const

Get dim property of the type, i.e. size of the static array.

get_dim(*typeinfo: VarInfo const implicit; index: int const*)

get_dim returns int

argument	argument type
typeinfo	*rtti::VarInfo* const implicit
index	int const

Get dim property of the type, i.e. size of the static array.

builtin_is_same_type (*a: rtti::TypeInfo const? const implicit; b: rtti::TypeInfo const? const implicit; refMatters: RefMatters const; cosntMatters: ConstMatters const; tempMatters: TemporaryMatters const; topLevel: bool const*)

builtin_is_same_type returns bool

argument	argument type
a	*rtti::TypeInfo* const? const implicit
b	*rtti::TypeInfo* const? const implicit
refMatters	*rtti::RefMatters* const
cosntMatters	*rtti::ConstMatters* const
tempMatters	*rtti::TemporaryMatters* const
topLevel	bool const

Returns true if two *TypeInfo* objects are the same given comparison criteria.

get_type_size (*type: rtti::TypeInfo? const implicit*)

get_type_size returns int

argument	argument type
type	*rtti::TypeInfo* ? const implicit

Returns size of the type in bytes.

get_type_align (*type: rtti::TypeInfo? const implicit*)

get_type_align returns int

argument	argument type
type	*rtti::TypeInfo* ? const implicit

Returns alignment of the type in bytes.

is_compatible_cast (*from: rtti::StructInfo const? const implicit; to: rtti::StructInfo const? const implicit*)

is_compatible_cast returns bool

argument	argument type
from	*rtti::StructInfo* const? const implicit
to	*rtti::StructInfo* const? const implicit

Returns true if *from* type can be casted to *to* type.

get_das_type_name (*type: Type const*)

get_das_type_name returns string

argument	argument type
type	*rtti::Type* const

Returns name of the *Type* object.

is_same_type (*a: TypeInfo const; b: TypeInfo const; refMatters: RefMatters const; constMatters: ConstMatters const; temporaryMatters: TemporaryMatters const; topLevel: bool const*)

is_same_type returns auto

argument	argument type
a	*rtti::TypeInfo* const
b	*rtti::TypeInfo* const
refMatters	*rtti::RefMatters* const
constMatters	*rtti::ConstMatters* const
temporaryMatters	*rtti::TemporaryMatters* const
topLevel	bool const

Returns true if two *TypeInfo* objects are the same given comparison criteria.

is_compatible_cast (*a: StructInfo const; b: StructInfo const*)

is_compatible_cast returns auto

argument	argument type
a	*rtti::StructInfo* const
b	*rtti::StructInfo* const

Returns true if *from* type can be casted to *to* type.

each_dim (*info: TypeInfo const*)

each_dim returns auto

argument	argument type
info	*rtti::TypeInfo* const

Iterates through all dim values of the rtti type object, i.e. through all size properties of the array.

each_dim (*info: VarInfo const*)

each_dim returns auto

argument	argument type
info	*rtti::VarInfo* const

Iterates through all dim values of the rtti type object, i.e. through all size properties of the array.

arg_types (*info: TypeInfo const*)

arg_types returns auto

argument	argument type
info	*rtti::TypeInfo* const

Iterates through argument types of the rtti type object.

arg_types (*info: VarInfo const*)

arg_types returns auto

argument	argument type
info	*rtti::VarInfo* const

Iterates through argument types of the rtti type object.

arg_names (*info: TypeInfo const*)

arg_names returns auto

argument	argument type
info	*rtti::TypeInfo* const

Iterates through argument names of the rtti type object.

arg_names (*info: VarInfo const*)

arg_names returns auto

argument	argument type
info	*rtti::VarInfo* const

Iterates through argument names of the rtti type object.

11.9 Rtti context access

- *get_total_functions (context:rtti::Context implicit) : int*
- *get_total_variables (context:rtti::Context implicit) : int*
- *get_function_info (context:any;index:int const) : rtti::FuncInfo const&*
- *get_variable_info (context:any;index:int const) : rtti::VarInfo const&*
- *get_variable_value (varInfo:rtti::VarInfo const implicit) : variant<tBool:bool;tInt:int;tUInt:uint;tInt64:int64;tUInt64:uint64;tFlo*
- *get_function_info (context:rtti::Context implicit;function:function<> const) : rtti::FuncInfo const? const*
- *get_function_by_mnh (context:rtti::Context implicit;MNH:uint64 const) : function<>*
- *get_line_info (line:__lineInfo const) : rtti::LineInfo*
- *get_line_info (depth:int const;context:__context const;line:__lineInfo const) : rtti::LineInfo*
- *this_context (context:__context const) : rtti::Context&*
- *context_for_each_function (blk:block<(info:rtti::FuncInfo const):void> const) : auto*
- *context_for_each_variable (blk:block<(info:rtti::VarInfo const):void> const) : auto*
- *class_info (cl:auto const) : rtti::StructInfo const?*
- *type_info (vinfo:rtti::LocalVariableInfo const) : rtti::TypeInfo const?*
- *type_info (vinfo:rtti::VarInfo const) : rtti::TypeInfo const?*

get_total_functions(*context: Context implicit*)

get_total_functions returns int

argument	argument type
context	*rtti::Context* implicit

Get total number of functions in the context.

get_total_variables(*context: Context implicit*)

get_total_variables returns int

argument	argument type
context	*rtti::Context* implicit

Get total number of global variables in the context.

get_function_info(*context: any; index: int const*)

get_function_info returns *rtti::FuncInfo* const&

argument	argument type
context	any
index	int const

Get function declaration info by index.

get_variable_info (*context: any; index: int const*)

get_variable_info returns *rtti::VarInfo* const&

argument	argument type
context	any
index	int const

Get global variable type information by variable index.

get_variable_value (*varInfo: VarInfo const implicit*)

get_variable_value returns *RttiValue*

argument	argument type
varInfo	*rtti::VarInfo* const implicit

Return RttiValue which represents value of the global variable.

get_function_info (*context: Context implicit; function: function<> const*)

get_function_info returns *rtti::FuncInfo* const? const

argument	argument type
context	*rtti::Context* implicit
function	function<> const

Get function declaration info by index.

get_function_by_mnh (*context: Context implicit; MNH: uint64 const*)

get_function_by_mnh returns function<>

argument	argument type
context	*rtti::Context* implicit
MNH	uint64 const

Returns *SimFunction* by mangled name hash.

get_line_info()

get_line_info returns *rtti::LineInfo*

Returns *LineInfo* object for the current line (line where get_line_info is called from).

get_line_info(*depth: int const*)

get_line_info returns *rtti::LineInfo*

argument	argument type
depth	int const

Returns *LineInfo* object for the current line (line where get_line_info is called from).

this_context()

this_context returns *rtti::Context* &

Returns current *Context* object.

context_for_each_function(*blk: block<(info:rtti::FuncInfo const):void> const*)

context_for_each_function returns auto

argument	argument type
blk	block<(info: *rtti::FuncInfo* const):void> const

Iterates through all functions in the *Context*.

context_for_each_variable(*blk: block<(info:rtti::VarInfo const):void> const*)

context_for_each_variable returns auto

argument	argument type
blk	block<(info: *rtti::VarInfo* const):void> const

Iterates through all variables in the *Context*.

class_info(*cl: auto const*)

class_info returns *rtti::StructInfo* const?

argument	argument type
cl	auto const

Returns *StructInfo?* ` for the class.

type_info (*vinfo: LocalVariableInfo const*)

type_info returns *rtti::TypeInfo* const?

argument	argument type
vinfo	*rtti::LocalVariableInfo* const

Returns *TypeInfo* object for the local variable.

type_info (*vinfo: VarInfo const*)

type_info returns *rtti::TypeInfo* const?

argument	argument type
vinfo	*rtti::VarInfo* const

Returns *TypeInfo* object for the local variable.

11.10 Program access

- *get_this_module (program:smart_ptr<rtti::Program> const implicit) : rtti::Module?*
- *get_module (name:string const implicit) : rtti::Module?*
- *program_for_each_module (program:smart_ptr<rtti::Program> const implicit;block:block<(var arg0:rtti::Module?):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *program_for_each_registered_module (block:block<(var arg0:rtti::Module?):void> const implicit;context:__context const;line:__lineInfo const) : void*

get_this_module (*program: smart_ptr<rtti::Program> const implicit*)

get_this_module returns *rtti::Module* ?

argument	argument type
program	smart_ptr< *rtti::Program* > const implicit

Get current *Program* object currently compiled module.

get_module (*name: string const implicit*) get_module returns *rtti::Module* ?

argument	argument type
name	string const implicit

Get *Module* object by name.

program_for_each_module (*program: smart_ptr<rtti::Program> const implicit; block: block<(var arg0:rtti::Module?):void> const implicit*)

argument	argument type
program	smart_ptr< *rtti::Program* > const implicit
block	block<(*rtti::Module* ?):void> const implicit

Iterates through all modules of the *Program* object.

program_for_each_registered_module (*block: block<(var arg0:rtti::Module?):void> const implicit*)

argument	argument type
block	block<(*rtti::Module* ?):void> const implicit

Iterates through all registered modules of the Daslang runtime.

11.11 Module access

- *module_for_each_structure (module:rtti::Module? const implicit;block:block<(arg0:rtti::StructInfo const):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *module_for_each_enumeration (module:rtti::Module? const implicit;block:block<(arg0:rtti::EnumInfo const):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *module_for_each_function (module:rtti::Module? const implicit;block:block<(arg0:rtti::FuncInfo const):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *module_for_each_generic (module:rtti::Module? const implicit;block:block<(arg0:rtti::FuncInfo const):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *module_for_each_global (module:rtti::Module? const implicit;block:block<(arg0:rtti::VarInfo const):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *module_for_each_annotation (module:rtti::Module? const implicit;block:block<(arg0:rtti::Annotation const):void> const implicit;context:__context const;line:__lineInfo const) : void*

module_for_each_structure (*module: rtti::Module? const implicit; block: block<(arg0:rtti::StructInfo const):void> const implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
block	block<(*rtti::StructInfo* const):void> const implicit

Iterates through all structure declarations in the *Module* object.

module_for_each_enumeration (*module: rtti::Module? const implicit; block: block<(arg0:rtti::EnumInfo const):void> const implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
block	block<(*rtti::EnumInfo* const):void> const implicit

Iterates through each enumeration in the module.

module_for_each_function (*module: rtti::Module? const implicit; block: block<(arg0:rtti::FuncInfo const):void> const implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
block	block<(*rtti::FuncInfo* const):void> const implicit

Iterates through each function in the module.

module_for_each_generic (*module: rtti::Module? const implicit; block: block<(arg0:rtti::FuncInfo const):void> const implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
block	block<(*rtti::FuncInfo* const):void> const implicit

Iterates through each generic function in the module.

module_for_each_global (*module: rtti::Module? const implicit; block: block<(arg0:rtti::VarInfo const):void> const implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
block	block<(*rtti::VarInfo* const):void> const implicit

Iterates through each global variable in the module.

module_for_each_annotation (*module: rtti::Module? const implicit; block: block<(arg0:rtti::Annotation const):void> const implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
block	block<(*rtti::Annotation* const):void> const implicit

Iterates though each handled type in the module.

11.12 Annotation access

- *get_annotation_argument_value (info:rtti::AnnotationArgument const implicit;context:__context const) : variant<tBool:bool;tInt:int;tUInt:uint;tInt64:int64;tUInt64:uint64;tFloat:float;tDouble:double;tString:string;nothing:any>*
- *add_annotation_argument (annotation:rtti::AnnotationArgumentList implicit;name:string const implicit) : int*

get_annotation_argument_value (*info: AnnotationArgument const implicit*)

get_annotation_argument_value returns *RttiValue*

argument	argument type
info	*rtti::AnnotationArgument* const implicit

Returns RttiValue which represents argument value for the specific annotation argument.

add_annotation_argument (*annotation: AnnotationArgumentList implicit; name: string const implicit*)

add_annotation_argument returns int

argument	argument type
annotation	*rtti::AnnotationArgumentList* implicit
name	string const implicit

Adds annotation argument to the *AnnotationArgumentList* object.

11.13 Compilation and simulation

- *compile (module_name:string const implicit;codeText:string const implicit;codeOfPolicies:rtti::CodeOfPolicies const implicit;block:block<(var arg0:bool;var arg1:smart_ptr<rtti::Program>;arg2:$::das_string const):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *compile (module_name:string const implicit;codeText:string const implicit;codeOfPolicies:rtti::CodeOfPolicies const implicit;exportAll:bool const;block:block<(var arg0:bool;var arg1:smart_ptr<rtti::Program>;arg2:$::das_string const):void> const implicit;context:__context const;line:__lineInfo const) : void*

- *compile_file (module_name:string const implicit;fileAccess:smart_ptr<rtti::FileAccess> const implicit;moduleGroup:rtti::ModuleGroup? const implicit;codeOfPolicies:rtti::CodeOfPolicies const implicit;block:block<(var arg0:bool;var arg1:smart_ptr<rtti::Program>;arg2:$::das_string const):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *for_each_expected_error (program:smart_ptr<rtti::Program> const implicit;block:block<(var arg0:rtti::CompilationError;var arg1:int):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *for_each_require_declaration (program:smart_ptr<rtti::Program> const implicit;block:block<(var arg0:rtti::Module?;arg1:string const#;arg2:string const#;var arg3:bool;arg4:rtti::LineInfo const&):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *simulate (program:smart_ptr<rtti::Program> const& implicit;block:block<(var arg0:bool;var arg1:smart_ptr<rtti::Context>;var arg2:$::das_string):void> const implicit;context:__context const;line:__lineInfo const) : void*

compile (*module_name: string const implicit; codeText: string const implicit; codeOfPolicies: CodeOfPolicies const implicit; block: block<(var arg0:bool;var arg1:smart_ptr<rtti::Program>;arg2:das_string const):void> const implicit*)

argument	argument type
module_name	string const implicit
codeText	string const implicit
codeOfPolicies	*rtti::CodeOfPolicies* const implicit
block	block<(bool;smart_ptr< *rtti::Program* >; *builtin::das_string* const):void> const implicit

Compile Daslang program given as string.

compile (*module_name: string const implicit; codeText: string const implicit; codeOfPolicies: CodeOfPolicies const implicit; exportAll: bool const; block: block<(var arg0:bool;var arg1:smart_ptr<rtti::Program>;arg2:das_string const):void> const implicit*)

argument	argument type
module_name	string const implicit
codeText	string const implicit
codeOfPolicies	*rtti::CodeOfPolicies* const implicit
exportAll	bool const
block	block<(bool;smart_ptr< *rtti::Program* >; *builtin::das_string* const):void> const implicit

Compile Daslang program given as string.

compile_file (*module_name: string const implicit; fileAccess: smart_ptr<rtti::FileAccess> const implicit; moduleGroup: rtti::ModuleGroup? const implicit; codeOfPolicies: CodeOfPolicies const implicit; block: block<(var arg0:bool;var arg1:smart_ptr<rtti::Program>;arg2:das_string const):void> const implicit*)

argument	argument type
module_name	string const implicit
fileAccess	smart_ptr< *rtti::FileAccess* > const implicit
moduleGroup	*rtti::ModuleGroup* ? const implicit
codeOfPolicies	*rtti::CodeOfPolicies* const implicit
block	block<(bool;smart_ptr< *rtti::Program* >; *builtin::das_string* const):void> const implicit

Compile Daslang program given as file in the *FileAccess* object.

for_each_expected_error (*program: smart_ptr<rtti::Program> const implicit; block: block<(var arg0:rtti::CompilationError;var arg1:int):void> const implicit*)

argument	argument type
program	smart_ptr< *rtti::Program* > const implicit
block	block<(*rtti::CompilationError* ;int):void> const implicit

Iterates through each compilation error of the *Program* object.

for_each_require_declaration (*program: smart_ptr<rtti::Program> const implicit; block: block<(var arg0:rtti::Module?;arg1:string const#;arg2:string const#;var arg3:bool;arg4:rtti::LineInfo const&):void> const implicit*)

argument	argument type
program	smart_ptr< *rtti::Program* > const implicit
block	block<(*rtti::Module* ?;string const#;string const#;bool; *rtti::LineInfo* const&):void> const implicit

Iterates though each *require* declaration of the compiled program.

simulate (*program: smart_ptr<rtti::Program> const& implicit; block: block<(var arg0:bool;var arg1:smart_ptr<rtti::Context>;var arg2:das_string):void> const implicit*)

argument	argument type
program	smart_ptr< *rtti::Program* > const& implicit
block	block<(bool;smart_ptr< *rtti::Context* >; *builtin::das_string*):void> const implicit

Simulates Daslang program and creates 'Context' object.

11.14 File access

- *make_file_access (project:string const implicit;context:__context const;at:__lineInfo const) : smart_ptr<rtti::FileAccess>*
- *set_file_source (access:smart_ptr<rtti::FileAccess> const implicit;fileName:string const implicit;text:string const implicit;context:__context const;line:__lineInfo const) : bool*
- *add_file_access_root (access:smart_ptr<rtti::FileAccess> const implicit;mod:string const implicit;path:string const implicit) : bool*

make_file_access (*project: string const implicit*)

make_file_access returns smart_ptr< *rtti::FileAccess* >

argument	argument type
project	string const implicit

Creates new *FileAccess* object.

set_file_source (*access: smart_ptr<rtti::FileAccess> const implicit; fileName: string const implicit; text: string const implicit*)

set_file_source returns bool

argument	argument type
access	smart_ptr< *rtti::FileAccess* > const implicit
fileName	string const implicit
text	string const implicit

Sets source for the specified file in the *FileAccess* object.

add_file_access_root (*access: smart_ptr<rtti::FileAccess> const implicit; mod: string const implicit; path: string const implicit*)

add_file_access_root returns bool

argument	argument type
access	smart_ptr< *rtti::FileAccess* > const implicit
mod	string const implicit
path	string const implicit

Add extra root directory (search path) to the *FileAccess* object.

11.15 Structure access

- *rtti_builtin_structure_for_each_annotation (struct:rtti::StructInfo const implicit;block:block<> const implicit;context:__context const;line:__lineInfo const) : void*
- *basic_struct_for_each_field (annotation:rtti::BasicStructureAnnotation const implicit;block:block<(var arg0:string;var arg1:string;arg2:rtti::TypeInfo const;var arg3:uint):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *basic_struct_for_each_parent (annotation:rtti::BasicStructureAnnotation const implicit;block:block<(var arg0:rtti::Annotation?):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *structure_for_each_annotation (st:rtti::StructInfo const;subexpr:block<(ann:rtti::Annotation const;args:rtti::AnnotationArguments const):void> const) : auto*

rtti_builtin_structure_for_each_annotation (*struct: StructInfo const implicit; block: block<> const implicit*)

argument	argument type
struct	*rtti::StructInfo* const implicit
block	block<> const implicit

Iterates through each annotation for the *Structure* object.

basic_struct_for_each_field (*annotation: BasicStructureAnnotation const implicit; block: block<(var arg0:string;var arg1:string;arg2:rtti::TypeInfo const;var arg3:uint):void> const implicit*)

argument	argument type
annotation	*rtti::BasicStructureAnnotation* const implicit
block	block<(string;string; *rtti::TypeInfo* const;uint):void> const implicit

Iterates through each field of the structure object.

basic_struct_for_each_parent (*annotation: BasicStructureAnnotation const implicit; block: block<(var arg0:rtti::Annotation?):void> const implicit*)

argument	argument type
annotation	*rtti::BasicStructureAnnotation* const implicit
block	block<(*rtti::Annotation* ?):void> const implicit

Iterates through each parent type of the *BasicStructureAnnotation* object.

structure_for_each_annotation (*st: StructInfo const; subexpr: block<(ann:rtti::Annotation const;args:rtti::AnnotationArguments const):void> const*)

structure_for_each_annotation returns auto

argument	argument type
st	*rtti::StructInfo* const
subexpr	block<(ann: *rtti::Annotation* const;args: *rtti::AnnotationArguments* const):void> const

Iterates through each annotation for the *Structure* object.

11.16 Data walking and printing

- *sprint_data (data:void? const implicit;type:rtti::TypeInfo const? const implicit;flags:bitfield const;context:__context const) : string*
- *sprint_data (data:float4 const;type:rtti::TypeInfo const? const implicit;flags:bitfield const;context:__context const) : string*
- *describe (type:rtti::TypeInfo const? const implicit;context:__context const) : string*
- *describe (lineinfo:rtti::LineInfo const implicit;fully:bool const;context:__context const) : string*
- *get_mangled_name (type:rtti::TypeInfo const? const implicit;context:__context const) : string*

sprint_data (*data: void? const implicit; type: rtti::TypeInfo const? const implicit; flags: bitfield const*)

sprint_data returns string

argument	argument type
data	void? const implicit
type	*rtti::TypeInfo* const? const implicit
flags	bitfield<> const

Prints data given *TypeInfo* and returns result as a string, similar to *print* function.

sprint_data (*data: float4 const; type: rtti::TypeInfo const? const implicit; flags: bitfield const*)

sprint_data returns string

argument	argument type
data	float4 const
type	*rtti::TypeInfo* const? const implicit
flags	bitfield<> const

Prints data given *TypeInfo* and returns result as a string, similar to *print* function.

describe (*type: rtti::TypeInfo const? const implicit*) describe returns string

argument	argument type
type	*rtti::TypeInfo* const? const implicit

Describe rtti object and return data as string.

describe (*lineinfo: LineInfo const implicit; fully: bool const*)

describe returns string

argument	argument type
lineinfo	*rtti::LineInfo* const implicit
fully	bool const

Describe rtti object and return data as string.

get_mangled_name (*type: rtti::TypeInfo const? const implicit*)

get_mangled_name returns string

argument	argument type
type	*rtti::TypeInfo* const? const implicit

Returns mangled name of the function.

11.17 Function and mangled name hash

- *get_function_by_mangled_name_hash (src:uint64 const;context:__context const) : function<>*
- *get_function_by_mangled_name_hash (src:uint64 const;context:rtti::Context implicit) : function<>*
- *get_function_mangled_name_hash (src:function<> const;context:__context const) : uint64*
- *get_function_address (MNH:uint64 const;at:rtti::Context implicit) : uint64*

get_function_by_mangled_name_hash (*src: uint64 const*)

get_function_by_mangled_name_hash returns function<>

argument	argument type
src	uint64 const

Returns *function*<> given mangled name hash.

get_function_by_mangled_name_hash (*src: uint64 const; context: Context implicit*) get_function_by_mangled_name_hash returns function<>

argument	argument type
src	uint64 const
context	*rtti::Context* implicit

Returns *function<>* given mangled name hash.

get_function_mangled_name_hash (*src: function<> const*)

get_function_mangled_name_hash returns uint64

argument	argument type
src	function<> const

Returns mangled name hash of the *function<>* object.

get_function_address (*MNH: uint64 const; at: Context implicit*)

get_function_address returns uint64

argument	argument type
MNH	uint64 const
at	*rtti::Context* implicit

Return function pointer *SimFunction* * given mangled name hash.

11.18 Context and mutex locking

- *lock_this_context (block:block<void> const implicit;context:__context const;line:__lineInfo const) : void*
- *lock_context (lock_context:rtti::Context implicit;block:block<void> const implicit;context:__context const;line:__lineInfo const) : void*
- *lock_mutex (mutex:rtti::recursive_mutex implicit;block:block<void> const implicit;context:__context const;line:__lineInfo const) : void*

lock_this_context (*block: block<void> const implicit*)

argument	argument type
block	block<> const implicit

Makes recursive critical section of the current *Context* object.

lock_context (*lock_context: Context implicit; block: block<void> const implicit*)

argument	argument type
lock_context	*rtti::Context* implicit
block	block<> const implicit

Makes recursive critical section of the given *Context* object.

lock_mutex (*mutex: recursive_mutex implicit; block: block<void> const implicit*)

argument	argument type
mutex	*rtti::recursive_mutex* implicit
block	block<> const implicit

Makes recursive critical section of the given recursive_mutex object.

11.19 Runtime data access

- *get_table_key_index (table:void? const implicit;key:any;baseType:rtti::Type const;valueTypeSize:int const;context:__context const;at:__lineInfo const) : int*

get_table_key_index (*table: void? const implicit; key: any; baseType: Type const; valueTypeSize: int const*)

get_table_key_index returns int

argument	argument type
table	void? const implicit
key	any
baseType	*rtti::Type* const
valueTypeSize	int const

Returns index of the key in the table.

11.20 Uncategorized

with_program_serialized (*program: block<(var arg0:smart_ptr<rtti::Program>):void> const implicit; block: smart_ptr<rtti::Program> const implicit*)

argument	argument type
program	block<(smart_ptr< *rtti::Program* >):void> const implicit
block	smart_ptr< *rtti::Program* > const implicit

Serializes program and then deserializes it. This is to test serialization.

get_tuple_field_offset (*type: rtti::TypeInfo? const implicit; index: int const*)

get_tuple_field_offset returns int

argument	argument type
type	*rtti::TypeInfo* ? const implicit
index	int const

Returns offset of the tuple field.

get_variant_field_offset (*type: rtti::TypeInfo? const implicit; index: int const*)

get_variant_field_offset returns int

argument	argument type
type	*rtti::TypeInfo* ? const implicit
index	int const

Returns offset of the variant field.

each (*info: FuncInfo implicit ==const*)

each returns iterator< *rtti::VarInfo* >

argument	argument type
info	*rtti::FuncInfo* implicit!

Iterates through each element of the object.

each (*info: FuncInfo const implicit ==const*)

each returns iterator< *rtti::VarInfo* const>

argument	argument type
info	*rtti::FuncInfo* const implicit!

Iterates through each element of the object.

each (*info: StructInfo implicit ==const*)

each returns iterator< *rtti::VarInfo* >

argument	argument type
info	*rtti::StructInfo* implicit!

Iterates through each element of the object.

each (*info: StructInfo const implicit ==const*)

each returns iterator< *rtti::VarInfo* const>

argument	argument type
info	*rtti::StructInfo* const implicit!

Iterates through each element of the object.

each (*info: EnumInfo implicit ==const*)

each returns iterator< *rtti::EnumValueInfo* >

argument	argument type
info	*rtti::EnumInfo* implicit!

Iterates through each element of the object.

each (*info: EnumInfo const implicit ==const*)

each returns iterator< *rtti::EnumValueInfo* const>

argument	argument type
info	*rtti::EnumInfo* const implicit!

Iterates through each element of the object.

CHAPTER

TWELVE

AST MANIPULATION LIBRARY

The AST module implements compilation time reflection for the Daslang syntax tree.

All functions and symbols are in “ast” module, use require to get access to it.

```
require ast
```

12.1 Type aliases

TypeDeclFlags is a bitfield

field	bit	value
ref	0	1
constant	1	2
temporary	2	4
_implicit	3	8
removeRef	4	16
removeConstant	5	32
removeDim	6	64
removeTemporary	7	128
explicitConst	8	256
aotAlias	9	512
smartPtr	10	1024
smartPtrNative	11	2048
isExplicit	12	4096

continues on next page

Table 1 – continued from previous page

field	bit	value
isNativeDim	13	8192
isTag	14	16384
explicitRef	15	32768

properties of the *TypeDecl* object.

FieldDeclarationFlags is a bitfield

field	bit	value
moveSemantics	0	1
parentType	1	2
capturedConstant	2	4
generated	3	8
capturedRef	4	16
doNotDelete	5	32
privateField	6	64
_sealed	7	128
implemented	8	256

properties of the *FieldDeclaration* object.

StructureFlags is a bitfield

field	bit	value
isClass	0	1
genCtor	1	2
cppLayout	2	4
cppLayoutNotPod	3	8
generated	4	16
persistent	5	32

continues on next page

Table 2 – continued from previous page

field	bit	value
isLambda	6	64
privateStructure	7	128
macroInterface	8	256
_sealed	9	512
skipLockCheck	10	1024
circular	11	2048
_generator	12	4096
hasStaticMembers	13	8192
hasStaticFunctions	14	16384

properties of the *Structure* object.

ExprGenFlags is a bitfield

field	bit	value
alwaysSafe	0	1
generated	1	2
userSaidItsSafe	2	4

generation (genFlags) properties of the *Expression* object.

ExprLetFlags is a bitfield

field	bit	value
inScope	0	1
hasEarlyOut	1	2

properties of the *ExprLet* object.

ExprFlags is a bitfield

field	bit	value
constexpression	0	1
noSideEffects	1	2
noNativeSideEffects	2	4
isForLoopSource	3	8
isCallArgument	4	16

properties of the *Expression* object.

ExprPrintFlags is a bitfield

field	bit	value
topLevel	0	1
argLevel	1	2
bottomLevel	2	4

printing properties of the *Expression* object.

FunctionFlags is a bitfield

field	bit	value
builtIn	0	1
policyBased	1	2
callBased	2	4
interopFn	3	8
hasReturn	4	16
copyOnReturn	5	32
moveOnReturn	6	64
exports	7	128
init	8	256
addr	9	512

continues on next page

Table 3 – continued from previous page

field	bit	value
used	10	1024
fastCall	11	2048
knownSideEffects	12	4096
hasToRunAtCompileTime	13	8192
unsafeOperation	14	16384
unsafeDeref	15	32768
hasMakeBlock	16	65536
aotNeedPrologue	17	131072
noAot	18	262144
aotHybrid	19	524288
aotTemplate	20	1048576
generated	21	2097152
privateFunction	22	4194304
_generator	23	8388608
_lambda	24	16777216
firstArgReturnType	25	33554432
noPointerCast	26	67108864
isClassMethod	27	134217728
isTypeConstructor	28	268435456
shutdown	29	536870912
anyTemplate	30	1073741824
macroInit	31	-2147483648

properties of the *Function* object.

`MoreFunctionFlags is a bitfield`

field	bit	value
macroFunction	0	1
needStringCast	1	2
aotHashDeppendsOnArguments	2	4
lateInit	3	8
requestJit	4	16
unsafeOutsideOfFor	5	32
skipLockCheck	6	64
safeImplicit	7	128
deprecated	8	256
aliasCMRES	9	512
neverAliasCMRES	10	1024
addressTaken	11	2048
propertyFunction	12	4096
pinvoke	13	8192
jitOnly	14	16384
isStaticClassMethod	15	32768

additional properties of the *Function* object.

FunctionSideEffectFlags is a bitfield

field	bit	value
_unsafe	0	1
userScenario	1	2
modifyExternal	2	4
modifyArgument	3	8
accessGlobal	4	16
invoke	5	32

side-effect properties of the *Function* object.

VariableFlags is a bitfield

field	bit	value
init_via_move	0	1
init_via_clone	1	2
used	2	4
aliasCMRES	3	8
marked_used	4	16
global_shared	5	32
do_not_delete	6	64
generated	7	128
capture_as_ref	8	256
can_shadow	9	512
private_variable	10	1024
tag	11	2048
global	12	4096
inScope	13	8192
no_capture	14	16384
early_out	15	32768
used_in_finally	16	65536
static_class_member	17	131072

properties of the *Variable* object.

VariableAccessFlags is a bitfield

field	bit	value
access_extern	0	1
access_get	1	2
access_ref	2	4
access_init	3	8
access_pass	4	16

access properties of the *Variable* object.

ExprBlockFlags is a bitfield

field	bit	value
isClosure	0	1
hasReturn	1	2
copyOnReturn	2	4
moveOnReturn	3	8
inTheLoop	4	16
finallyBeforeBody	5	32
finallyDisabled	6	64
aotSkipMakeBlock	7	128
aotDoNotSkipAnnotationData	8	256
isCollapseable	9	512
needCollapse	10	1024
hasMakeBlock	11	2048
hasEarlyOut	12	4096

properties of the *ExrpBlock* object.

ExprAtFlags is a bitfield

field	bit	value
r2v	0	1
r2cr	1	2
write	2	4
no_promotion	3	8

properties of the *ExprAt* object.

`ExprMakeLocalFlags is a bitfield`

field	bit	value
useStackRef	0	1
useCMRES	1	2
doesNotNeedSp	2	4
doesNotNeedInit	3	8
initAllFields	4	16
alwaysAlias	5	32

properties of the *ExprMakeLocal* object (*ExprMakeArray*, *ExprMakeStruct*, 'ExprMakeTuple', 'ExprMakeVariant').

`ExprAscendFlags is a bitfield`

field	bit	value
useStackRef	0	1
needTypeInfo	1	2
isMakeLambda	2	4

properties of the *ExprAscend* object.

`ExprCastFlags is a bitfield`

field	bit	value
upcastCast	0	1
reinterpretCast	1	2

properties of the *ExprCast* object.

ExprVarFlags is a bitfield

field	bit	value
local	0	1
argument	1	2
_block	2	4
thisBlock	3	8
r2v	4	16
r2cr	5	32
write	6	64

properties of the *ExprVar* object.

ExprMakeStructFlags is a bitfield

field	bit	value
useInitializer	0	1
isNewHandle	1	2

properties of the *ExprMakeStruct* object.

MakeFieldDeclFlags is a bitfield

field	bit	value
moveSemantics	0	1
cloneSemantics	1	2

properties of the *MakeFieldDecl* object.

ExprFieldDerefFlags is a bitfield

field	bit	value
unsafeDeref	0	1
ignoreCaptureConst	1	2

dereferencing properties of the *ExprField* object.

ExprFieldFieldFlags is a bitfield

field	bit	value
r2v	0	1
r2cr	1	2
write	2	4
no_promotion	3	8

field properties of the *ExprField* object.

ExprSwizzleFieldFlags is a bitfield

field	bit	value
r2v	0	1
r2cr	1	2
write	2	4

properties of the *ExprSwizzle* object.

ExprYieldFlags is a bitfield

field	bit	value
moveSemantics	0	1
skipLockCheck	1	2

properties of the *ExprYield* object.

ExprReturnFlags is a bitfield

field	bit	value
moveSemantics	0	1
returnReference	1	2
returnInBlock	2	4
takeOverRightStack	3	8
returnCallCMRES	4	16
returnCMRES	5	32
fromYield	6	64
fromComprehension	7	128
skipLockCheck	8	256

properties of the *ExprReturn* object.

ExprMakeBlockFlags is a bitfield

field	bit	value
isLambda	0	1
isLocalFunction	1	2

properties of the *ExprMakeBlock* object.

CopyFlags is a bitfield

field	bit	value
allowCopyTemp	0	1
takeOverRightStack	1	2

properties of the *ExprCopy* object.

MoveFlags is a bitfield

field	bit	value
skipLockCheck	0	1
takeOverRightStack	1	2

properties of the *ExprMove* object.

IfFlags is a bitfield

field	bit	value
isStatic	0	1
doNotFold	1	2

properties of the *ExprIf* object.

ExpressionPtr = smart_ptr<ast::Expression>

Smart pointer to *Expression* object.

ProgramPtr = smart_ptr<rtti::Program>

Smart pointer to *Program* object.

TypeDeclPtr = smart_ptr<ast::TypeDecl>

Smart pointer to *TypeDecl* object.

VectorTypeDeclPtr = dasvector`smart_ptr`TypeDecl

Smart pointer to das::vector<ExpressionPtr>.

EnumerationPtr = smart_ptr<ast::Enumeration>

Smart pointer to *Enumeration* object.

StructurePtr = smart_ptr<ast::Structure>

Smart pointer to *Structure* object.

FunctionPtr = smart_ptr<ast::Function>

Smart pointer to *Function* object.

VariablePtr = smart_ptr<ast::Variable>

Smart pointer to *Variable* object.

MakeFieldDeclPtr = smart_ptr<ast::MakeFieldDecl>

Smart pointer to *MakeFieldDecl* object.

FunctionAnnotationPtr = smart_ptr<ast::FunctionAnnotation>

Smart pointer to *FunctionAnnotation* object.

StructureAnnotationPtr = smart_ptr<ast::StructureAnnotation>

Smart pointer to *StructureAnnotation* object.

EnumerationAnnotationPtr = smart_ptr<ast::EnumerationAnnotation>

Smart pointer to *EnumerationAnnotation* object.

PassMacroPtr = smart_ptr<ast::PassMacro>

Smart pointer to *PassMacro* object.

VariantMacroPtr = smart_ptr<ast::VariantMacro>

Smart pointer to *VariantMacro* object.

ReaderMacroPtr = smart_ptr<ast::ReaderMacro>

Smart pointer to *ReaderMacro* object.

CommentReaderPtr = smart_ptr<ast::CommentReader>

Smart pointer to *CommentReader* object.

CallMacroPtr = smart_ptr<ast::CallMacro>

Smart pointer to *CallMacro* object.

TypeInfoMacroPtr = smart_ptr<ast::TypeInfoMacro>

Smart pointer to *TypeInfoMacro* object.

ForLoopMacroPtr = smart_ptr<ast::ForLoopMacro>

Smart pointer to 'ForLoopMacro'.

CaptureMacroPtr = smart_ptr<ast::CaptureMacro>

Smart pointer to 'CaptureMacro'.

SimulateMacroPtr = smart_ptr<ast::SimulateMacro>

Smart pointer to *SimulateMacro* object.

12.2 Enumerations

SideEffects

none	0
unsafe	1
userScenario	2
modifyExternal	4
accessExternal	4
modifyArgument	8
modifyArgumentAndExternal	12
worstDefault	12
accessGlobal	16
invoke	32
inferredSideEffects	56

Enumeration with all possible side effects of expression or function.

CaptureMode

capture_any	0
capture_by_copy	1
capture_by_reference	2
capture_by_clone	3
capture_by_move	4

Enumeration with lambda variables capture modes.

12.3 Handled structures

`ModuleLibrary`

Object which holds list of *Module* and provides access to them.

`Expression`

Expression fields are

at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Any expression (base class).

`TypeDecl`

TypeDecl fields are

alias	*builtin::das_string*
annotation	*rtti::TypeAnnotation* ?
dimExpr	vector<smart_ptr<Expression>>
argTypes	vector<smart_ptr<TypeDecl>>
dim	vector<int>
_module	*rtti::Module* ?
secondType	smart_ptr< *ast::TypeDecl* >
at	*rtti::LineInfo*
enumType	*ast::Enumeration* ?
argNames	vector<das_string>
baseType	*rtti::Type*
firstType	smart_ptr< *ast::TypeDecl* >
structType	*ast::Structure* ?
flags	*TypeDeclFlags*

TypeDecl property operators are

canAot	bool
isExprType	bool
isSimpleType	bool
isArray	bool
isGoodIteratorType	bool
isGoodArrayType	bool
isGoodTableType	bool
isGoodBlockType	bool
isGoodFunctionType	bool
isGoodLambdaType	bool

continues on next page

Table 6 – continued from previous page

isGoodTupleType	bool
isGoodVariantType	bool
isVoid	bool
isRef	bool
isRefType	bool
canWrite	bool
isAotAlias	bool
isShareable	bool
isIndex	bool
isBool	bool
isInteger	bool
isSignedInteger	bool
isUnsignedInteger	bool
isSignedIntegerOrIntVec	bool
isUnsignedIntegerOrIntVec	bool
isFloatOrDouble	bool
isNumeric	bool
isNumericComparable	bool
isPointer	bool
isVoidPointer	bool
isIterator	bool
isEnum	bool
isEnumT	bool
isHandle	bool
isStructure	bool
isClass	bool

continues on next page

Table 6 – continued from previous page

isFunction	bool
isTuple	bool
isVariant	bool
sizeOf	int
countOf	int
alignOf	int
baseSizeOf	int
stride	int
tupleSize	int
tupleAlign	int
variantSize	int
variantAlign	int
canCopy	bool
canMove	bool
canClone	bool
canNew	bool
canDeletePtr	bool
canDelete	bool
needDelete	bool
isPod	bool
isRawPod	bool
isNoHeapType	bool
isWorkhorseType	bool
isPolicyType	bool
isVecPolicyType	bool

continues on next page

Table 6 – continued from previous page

isReturnType	bool
isCtorType	bool
isRange	bool
isString	bool
isConst	bool
isFoldable	bool
isAlias	bool
isAutoArrayResolved	bool
isAuto	bool
isAutoOrAlias	bool
isVectorType	bool
isBitfield	bool
isLocal	bool
hasClasses	bool
hasNonTrivialCtor	bool
hasNonTrivialDtor	bool
hasNonTrivialCopy	bool
canBePlacedInContainer	bool
vectorBaseType	*rtti::Type*
vectorDim	int
canInitWithZero	bool
rangeBaseType	*rtti::Type*

Any type declaration.

Structure

Structure fields are

_module	*rtti::Module* ?
at	*rtti::LineInfo*
parent	*ast::Structure* ?
annotations	*rtti::AnnotationList*
name	*builtin::das_string*
fields	vector<FieldDeclaration>
flags	*StructureFlags*

Structure declaration.

FieldDeclaration

FieldDeclaration fields are

annotation	*rtti::AnnotationArgumentList*
at	*rtti::LineInfo*
name	*builtin::das_string*
init	smart_ptr< *ast::Expression* >
offset	int
_type	smart_ptr< *ast::TypeDecl* >
flags	*FieldDeclarationFlags*

Structure field declaration.

EnumEntry

EnumEntry fields are

value	smart_ptr< *ast::Expression* >
at	*rtti::LineInfo*
name	*builtin::das_string*

Entry in the enumeration.

Enumeration

Enumeration fields are

_module	*rtti::Module* ?
at	*rtti::LineInfo*
isPrivate	bool
cppName	*builtin::das_string*
list	vector<EnumEntry>
annotations	*rtti::AnnotationList*
name	*builtin::das_string*
external	bool
baseType	*rtti::Type*

Enumeration declaration.

Function

Function fields are

arguments	vector<smart_ptr<Variable>>
fromGeneric	smart_ptr< *ast::Function* >
result	smart_ptr< *ast::TypeDecl* >
aotHash	uint64
totalGenLabel	int
_module	*rtti::Module* ?
index	int
at	*rtti::LineInfo*
inferStack	vector<InferHistory>
body	smart_ptr< *ast::Expression* >
atDecl	*rtti::LineInfo*
sideEffectFlags	*FunctionSideEffectFlags*
annotations	*rtti::AnnotationList*
totalStackSize	uint

continues on next page

Table 7 – continued from previous page

name	*builtin::das_string*
moreFlags	*MoreFunctionFlags*
hash	uint64
classParent	*ast::Structure* ?
flags	*FunctionFlags*

Function property operators are

origin	smart_ptr< *ast::Function* >
isGeneric	bool

Function declaration.

InferHistory

InferHistory fields are

func	*ast::Function* ?
at	*rtti::LineInfo*

Generic function infer history. Contains stack on where the function was first instantiated from (*Function* and *LineInfo* pairs).

Variable

Variable fields are

annotation	*rtti::AnnotationArgumentList*
initStackSize	uint
_module	*rtti::Module* ?
index	int
at	*rtti::LineInfo*
stackTop	uint
name	*builtin::das_string*
init	smart_ptr< *ast::Expression* >
_aka	*builtin::das_string*
access_flags	*VariableAccessFlags*
source	smart_ptr< *ast::Expression* >
_type	smart_ptr< *ast::TypeDecl* >
flags	*VariableFlags*

Variable property operators are

isAccessUnused	bool

Variable declaration.

AstContext

AstContext fields are

func	smart_ptr< *ast::Function* >
scopes	vector<smart_ptr<Expression>>
blocks	vector<smart_ptr<Expression>>
_loop	vector<smart_ptr<Expression>>
_with	vector<smart_ptr<Expression>>

Lexical context for the particular expression. Contains current function, loops, blocks, scopes, and with sections.

ExprBlock

ExprBlock fields are

stackVarBottom	uint
annotationDataSid	uint64
arguments	vector<smart_ptr<Variable>>
at	*rtti::LineInfo*
stackCleanVars	vector<pair`uint`uint>
list	vector<smart_ptr<Expression>>
returnType	smart_ptr< *ast::TypeDecl* >
printFlags	*ExprPrintFlags*
annotations	*rtti::AnnotationList*
stackTop	uint
maxLabelIndex	int
blockFlags	*ExprBlockFlags*
finalList	vector<smart_ptr<Expression>>
genFlags	*ExprGenFlags*
annotationData	uint64
stackVarTop	uint
flags	*ExprFlags*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const

Any block expression, including regular blocks and all types of closures. For the closures block arguments are defined. Finally section is defined, if exists.

ExprLet

ExprLet fields are

atInit	*rtti::LineInfo*
at	*rtti::LineInfo*
letFlags	*ExprLetFlags*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
variables	vector<smart_ptr<Variable>>
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Local variable declaration (*let v = expr;*).

ExprStringBuilder

ExprStringBuilder fields are

at	*rtti::LineInfo*
elements	vector<smart_ptr<Expression>>
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

String builder expression ("blah{blah1}blah2").

MakeFieldDecl

MakeFieldDecl fields are

value	smart_ptr< *ast::Expression* >
at	*rtti::LineInfo*
name	*builtin::das_string*
tag	smart_ptr< *ast::Expression* >
flags	*MakeFieldDeclFlags*

Part of *ExprMakeStruct*, declares single field (*a* = *expr* or *a* <- *expr* etc)

ExprNamedCall

ExprNamedCall fields are

arguments	*ast::MakeStruct*
at	*rtti::LineInfo*
nonNamedArguments	vector<smart_ptr<Expression>>
printFlags	*ExprPrintFlags*
name	*builtin::das_string*
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Named call (*call([argname1=expr1, argname2=expr2])*).

ExprLooksLikeCall

ExprLooksLikeCall fields are

atEnclosure	*rtti::LineInfo*
arguments	vector<smart_ptr<Expression>>
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
name	*builtin::das_string*
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Anything which looks like call (*call(expr1,expr2)*).

ExprCallFunc

ExprCallFunc fields are

atEnclosure	*rtti::LineInfo*
func	*ast::Function* ?
arguments	vector<smart_ptr<Expression>>
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
stackTop	uint
name	*builtin::das_string*
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Actual function call (*func(expr1,. . .)*).

ExprNew

ExprNew fields are

atEnclosure	*rtti::LineInfo*
func	*ast::Function* ?
typeexpr	smart_ptr< *ast::TypeDecl* >
arguments	vector<smart_ptr<Expression>>
at	*rtti::LineInfo*
initializer	bool
printFlags	*ExprPrintFlags*
stackTop	uint
name	*builtin::das_string*
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

New expression (*new Foo*, *new Bar(expr1..)*, but **NOT** *new [[Foo …]]*)

ExprCall

ExprCall fields are

atEnclosure	*rtti::LineInfo*
func	*ast::Function* ?
arguments	vector<smart_ptr<Expression>>
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
stackTop	uint
name	*builtin::das_string*
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
cmresAlias	bool
doesNotNeedSp	bool
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Anything which looks like call (*call(expr1,expr2)*).

ExprPtr2Ref

ExprPtr2Ref fields are

at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
subexpr	smart_ptr< *ast::Expression* >
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Pointer dereference (**expr* or *deref(expr)*).

ExprNullCoalescing

ExprNullCoalescing fields are

defaultValue	smart_ptr< *ast::Expression* >
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
subexpr	smart_ptr< *ast::Expression* >
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Null coalescing (*expr1 ?? expr2*).

ExprAt

ExprAt fields are

index	smart_ptr< *ast::Expression* >
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
subexpr	smart_ptr< *ast::Expression* >
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const
atFlags	*ExprAtFlags*

Index lookup (*expr[expr1]*).

ExprSafeAt

ExprSafeAt fields are

index	smart_ptr< *ast::Expression* >
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
subexpr	smart_ptr< *ast::Expression* >
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const
atFlags	*ExprAtFlags*

Safe index lookup (*expr?[expr1]*).

ExprIs

ExprIs fields are

typeexpr	smart_ptr< *ast::TypeDecl* >
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
subexpr	smart_ptr< *ast::Expression* >
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Is expression for variants and such (*expr is Foo*).

ExprOp

Compilation time only base class for any operator.

ExprOp2

ExprOp2 fields are

atEnclosure	*rtti::LineInfo*
func	*ast::Function* ?
arguments	vector<smart_ptr<Expression>>
right	smart_ptr< *ast::Expression* >
at	*rtti::LineInfo*
op	*builtin::das_string*
printFlags	*ExprPrintFlags*
stackTop	uint
name	*builtin::das_string*
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const
left	smart_ptr< *ast::Expression* >

Two operand operator (*expr1* + *expr2*)

ExprOp3

ExprOp3 fields are

atEnclosure	*rtti::LineInfo*
func	*ast::Function* ?
arguments	vector<smart_ptr<Expression>>
right	smart_ptr< *ast::Expression* >
at	*rtti::LineInfo*
op	*builtin::das_string*
printFlags	*ExprPrintFlags*
stackTop	uint

continues on next page

Table 9 – continued from previous page

name	*builtin::das_string*
subexpr	smart_ptr< *ast::Expression* >
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const
left	smart_ptr< *ast::Expression* >

Three operand operator (*cond ? expr1 : expr2*)

ExprCopy

ExprCopy fields are

atEnclosure	*rtti::LineInfo*
func	*ast::Function* ?
arguments	vector<smart_ptr<Expression>>
right	smart_ptr< *ast::Expression* >
at	*rtti::LineInfo*
copy_flags	*CopyFlags*
op	*builtin::das_string*
printFlags	*ExprPrintFlags*
stackTop	uint
name	*builtin::das_string*
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*

continues on next page

Table 10 – continued from previous page

__rtti	string const
left	smart_ptr< *ast::Expression* >

Copy operator (*expr1 = expr2*)

ExprMove

ExprMove fields are

atEnclosure	*rtti::LineInfo*
func	*ast::Function* ?
arguments	vector<smart_ptr<Expression>>
right	smart_ptr< *ast::Expression* >
at	*rtti::LineInfo*
op	*builtin::das_string*
move_flags	*MoveFlags*
printFlags	*ExprPrintFlags*
stackTop	uint
name	*builtin::das_string*
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const
left	smart_ptr< *ast::Expression* >

Move operator (*expr1 <- expr2*)

ExprClone

ExprClone fields are

atEnclosure	*rtti::LineInfo*
func	*ast::Function* ?
arguments	vector<smart_ptr<Expression>>
right	smart_ptr< *ast::Expression* >
at	*rtti::LineInfo*
op	*builtin::das_string*
printFlags	*ExprPrintFlags*
stackTop	uint
name	*builtin::das_string*
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const
left	smart_ptr< *ast::Expression* >

Clone operator (*expr1 := expr2*)

ExprWith

ExprWith fields are

at	*rtti::LineInfo*
body	smart_ptr< *ast::Expression* >
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
_with	smart_ptr< *ast::Expression* >
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

With section (*with expr {your; block; here}*).

ExprAssume

ExprAssume fields are

alias	*builtin::das_string*
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
subexpr	smart_ptr< *ast::Expression* >
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Assume expression (*assume name = expr*).

ExprWhile

ExprWhile fields are

at	*rtti::LineInfo*
body	smart_ptr< *ast::Expression* >
cond	smart_ptr< *ast::Expression* >
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

While loop (*while expr {your; block; here;}*)

ExprTryCatch

ExprTryCatch fields are

try_block	smart_ptr< *ast::Expression* >
at	*rtti::LineInfo*
catch_block	smart_ptr< *ast::Expression* >
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Try-recover expression (*try {your; block; here;} recover {your; recover; here;}*)

ExprIfThenElse

ExprIfThenElse fields are

if_flags	*IfFlags*
at	*rtti::LineInfo*
if_false	smart_ptr< *ast::Expression* >
cond	smart_ptr< *ast::Expression* >
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
if_true	smart_ptr< *ast::Expression* >
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

If-then-else expression (*if expr1 {your; block; here;} else {your; block; here;}*) including *static_if*'s.

ExprFor

ExprFor fields are

visibility	*rtti::LineInfo*

continues on next page

Table 12 – continued from previous page

allowIteratorOptimization	bool
canShadow	bool
iteratorsTags	vector<smart_ptr<Expression>>
at	*rtti::LineInfo*
body	smart_ptr< *ast::Expression* >
iteratorsAt	vector<LineInfo>
printFlags	*ExprPrintFlags*
iterators	vector<das_string>
iteratorVariables	vector<smart_ptr<Variable>>
genFlags	*ExprGenFlags*
iteratorsAka	vector<das_string>
sources	vector<smart_ptr<Expression>>
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

For loop (*for expr1 in expr2 {your; block; here;}*)

ExprMakeLocal

ExprMakeLocal fields are

makeType	smart_ptr< *ast::TypeDecl* >
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
makeFlags	*ExprMakeLocalFlags*
stackTop	uint
extraOffset	uint
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Any make expression (*ExprMakeBlock*, *ExprMakeTuple*, *ExprMakeVariant*, *ExprMakeStruct*)

ExprMakeStruct

ExprMakeStruct fields are

makeType	smart_ptr< *ast::TypeDecl* >
at	*rtti::LineInfo*
structs	vector<smart_ptr<MakeStruct>>
printFlags	*ExprPrintFlags*
makeFlags	*ExprMakeLocalFlags*
stackTop	uint
extraOffset	uint
genFlags	*ExprGenFlags*
_block	smart_ptr< *ast::Expression* >
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const
makeStructFlags	*ExprMakeStructFlags*

Make structure expression (*[[YourStruct v1=expr1elem1, v2=expr2elem1, . . . ; v1=expr1elem2, . . .]]*)

ExprMakeVariant

ExprMakeVariant fields are

variants	vector<smart_ptr<MakeFieldDecl>>
makeType	smart_ptr< *ast::TypeDecl* >
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
makeFlags	*ExprMakeLocalFlags*
stackTop	uint
extraOffset	uint
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Make variant expression (*[YourVariant variantName=expr1]*)

ExprMakeArray

ExprMakeArray fields are

makeType	smart_ptr< *ast::TypeDecl* >
values	vector<smart_ptr<Expression>>
at	*rtti::LineInfo*
recordType	smart_ptr< *ast::TypeDecl* >
printFlags	*ExprPrintFlags*
makeFlags	*ExprMakeLocalFlags*
stackTop	uint
extraOffset	uint
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Make array expression (*[[auto 1;2;3]]* or *[{auto "foo";"bar"}]* for static and dynamic arrays accordingly).

ExprMakeTuple

ExprMakeTuple fields are

makeType	smart_ptr< *ast::TypeDecl* >
values	vector<smart_ptr<Expression>>
at	*rtti::LineInfo*
recordType	smart_ptr< *ast::TypeDecl* >
printFlags	*ExprPrintFlags*
makeFlags	*ExprMakeLocalFlags*
stackTop	uint
extraOffset	uint
genFlags	*ExprGenFlags*
isKeyValue	bool
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Make tuple expression (*[[auto f1,f2,f3]]*)

ExprArrayComprehension

ExprArrayComprehension fields are

at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
generatorSyntax	bool
subexpr	smart_ptr< *ast::Expression* >
genFlags	*ExprGenFlags*
exprFor	smart_ptr< *ast::Expression* >
exprWhere	smart_ptr< *ast::Expression* >
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Array comprehension (*[{for x in 0..3; x}]*, *[[for y in range(100); x*2; where x!=13]]* for arrays or generators accordingly).

TypeInfoMacro

TypeInfoMacro fields are

_module	*rtti::Module* ?
name	*builtin::das_string*

Compilation time only structure which holds live information about typeinfo expression for the specific macro.

ExprTypeInfo

ExprTypeInfo fields are

typeexpr	smart_ptr< *ast::TypeDecl* >
extratrait	*builtin::das_string*
macro	*ast::TypeInfoMacro* ?
subtrait	*builtin::das_string*
at	*rtti::LineInfo*
trait	*builtin::das_string*
printFlags	*ExprPrintFlags*
subexpr	smart_ptr< *ast::Expression* >
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

typeinfo() expression (*typeinfo(dim a)*, *typeinfo(is_ref_type type<int&>)*)

ExprTypeDecl

ExprTypeDecl fields are

typeexpr	smart_ptr< *ast::TypeDecl* >
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

typedecl() expression (*typedecl(1+2)*)

ExprLabel

ExprLabel fields are

comment	*builtin::das_string*
at	*rtti::LineInfo*
labelName	int
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Label (*label 13:*)

ExprGoto

ExprGoto fields are

at	*rtti::LineInfo*
labelName	int
printFlags	*ExprPrintFlags*
subexpr	smart_ptr< *ast::Expression* >
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Goto expression (*goto label 13*, *goto x*)

ExprRef2Value

ExprRef2Value fields are

at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
subexpr	smart_ptr< *ast::Expression* >
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Compilation time only structure which holds reference to value conversion for the value types, i.e. goes from int& to int and such.

ExprRef2Ptr

ExprRef2Ptr fields are

at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
subexpr	smart_ptr< *ast::Expression* >
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Addr expresion (*addr(expr)*)

ExprAddr

ExprAddr fields are

func	*ast::Function* ?
target	*builtin::das_string*
at	*rtti::LineInfo*
funcType	smart_ptr< *ast::TypeDecl* >
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Function address (@@*foobarfunc* or @@*foobarfunc<(int;int):bool>*)

ExprAssert

ExprAssert fields are

atEnclosure	*rtti::LineInfo*
arguments	vector<smart_ptr<Expression>>
isVerify	bool
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
name	*builtin::das_string*
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Assert expression (*assert(x<13)* or *assert(x<13, "x is too big")*)

ExprQuote

ExprQuote fields are

atEnclosure	*rtti::LineInfo*
arguments	vector<smart_ptr<Expression>>
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
name	*builtin::das_string*
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Compilation time expression which holds its subexpressions but does not infer them (*quote()* <| *x+5*)

ExprStaticAssert

ExprStaticAssert fields are

atEnclosure	*rtti::LineInfo*
arguments	vector<smart_ptr<Expression>>
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
name	*builtin::das_string*
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Static assert expression (*static_assert(x<13)* or *static_assert(x<13, "x is too big")*)

ExprDebug

ExprDebug fields are

atEnclosure	*rtti::LineInfo*
arguments	vector<smart_ptr<Expression>>
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
name	*builtin::das_string*
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Debug expression (*debug(x)* or *debug(x,"x=")*)

ExprInvoke

ExprInvoke fields are

atEnclosure	*rtti::LineInfo*
arguments	vector<smart_ptr<Expression>>
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
stackTop	uint
name	*builtin::das_string*
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
isInvokeMethod	bool
doesNotNeedSp	bool
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Invoke expression (*invoke(fn)* or *invoke(lamb, arg1, arg2, …)*)

`ExprErase`

ExprErase fields are

atEnclosure	*rtti::LineInfo*
arguments	vector<smart_ptr<Expression>>
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
name	*builtin::das_string*
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Erase expression (*erase(tab,key)*)

`ExprSetInsert`

ExprSetInsert fields are

atEnclosure	*rtti::LineInfo*
arguments	vector<smart_ptr<Expression>>
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
name	*builtin::das_string*
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

insert(tab, at) for the table<keyType; void> aka table<keyType>

`ExprFind`

ExprFind fields are

atEnclosure	*rtti::LineInfo*
arguments	vector<smart_ptr<Expression>>
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
name	*builtin::das_string*
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Find expression (*find(tab,key) <| { your; block; here; }*)

ExprKeyExists

ExprKeyExists fields are

atEnclosure	*rtti::LineInfo*
arguments	vector<smart_ptr<Expression>>
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
name	*builtin::das_string*
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Key exists expression (*key_exists(tab,key)*)

ExprAscend

ExprAscend fields are

ascType	smart_ptr< *ast::TypeDecl* >
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
stackTop	uint
ascendFlags	*ExprAscendFlags*
subexpr	smart_ptr< *ast::Expression* >
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

New expression for ExprMakeLocal (*new [[Foo fld=val,. . .]]* or *new [[Foo() fld=. . .]]*, but **NOT** *new Foo()*)

`ExprCast`

ExprCast fields are

castFlags	*ExprCastFlags*
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
subexpr	smart_ptr< *ast::Expression* >
castType	smart_ptr< *ast::TypeDecl* >
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Any cast expression (*cast<int> a*, *upcast<Foo> b* or *reinterpret<Bar?> c*)

`ExprDelete`

ExprDelete fields are

at	*rtti::LineInfo*
native	bool
printFlags	*ExprPrintFlags*
subexpr	smart_ptr< *ast::Expression* >
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Delete expression (*delete blah*)

ExprVar

ExprVar fields are

at	*rtti::LineInfo*
variable	smart_ptr< *ast::Variable* >
varFlags	*ExprVarFlags*
printFlags	*ExprPrintFlags*
argumentIndex	int
name	*builtin::das_string*
genFlags	*ExprGenFlags*
pBlock	*ast::ExprBlock* ?
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Variable access (*foo*)

ExprTag

ExprTag fields are

value	smart_ptr< *ast::Expression* >
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
subexpr	smart_ptr< *ast::Expression* >
name	*builtin::das_string*
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Compilation time only tag expression, used for reification. For example $c(....).

ExprSwizzle

ExprSwizzle fields are

value	smart_ptr< *ast::Expression* >
at	*rtti::LineInfo*
fieldFlags	*ExprSwizzleFieldFlags*
mask	*builtin::das_string*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const
fields	vector<uint8>

Vector swizzle operatrion (*vec.xxy* or *vec.y*)

ExprField

ExprField fields are

annotation	smart_ptr< *rtti::TypeAnnotation* >
value	smart_ptr< *ast::Expression* >
at	*rtti::LineInfo*
fieldIndex	int
fieldFlags	*ExprFieldFieldFlags*
field	*ast::FieldDeclaration* const? const
derefFlags	*ExprFieldDerefFlags*
printFlags	*ExprPrintFlags*
name	*builtin::das_string*
atField	*rtti::LineInfo*
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Field lookup (*foo.bar*)

ExprSafeField

ExprSafeField fields are

annotation	smart_ptr< *rtti::TypeAnnotation* >
value	smart_ptr< *ast::Expression* >
at	*rtti::LineInfo*
fieldIndex	int
fieldFlags	*ExprFieldFieldFlags*
field	*ast::FieldDeclaration* const? const
skipQQ	bool
derefFlags	*ExprFieldDerefFlags*
printFlags	*ExprPrintFlags*
name	*builtin::das_string*
atField	*rtti::LineInfo*
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Safe field lookup (*foo?.bar*)

ExprIsVariant

ExprIsVariant fields are

annotation	smart_ptr< *rtti::TypeAnnotation* >
value	smart_ptr< *ast::Expression* >
at	*rtti::LineInfo*
fieldIndex	int
fieldFlags	*ExprFieldFieldFlags*
field	*ast::FieldDeclaration* const? const
derefFlags	*ExprFieldDerefFlags*
printFlags	*ExprPrintFlags*
name	*builtin::das_string*
atField	*rtti::LineInfo*
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Is expression (*foo is bar*)

ExprAsVariant

ExprAsVariant fields are

annotation	smart_ptr< *rtti::TypeAnnotation* >
value	smart_ptr< *ast::Expression* >
at	*rtti::LineInfo*
fieldIndex	int
fieldFlags	*ExprFieldFieldFlags*
field	*ast::FieldDeclaration* const? const
derefFlags	*ExprFieldDerefFlags*
printFlags	*ExprPrintFlags*
name	*builtin::das_string*
atField	*rtti::LineInfo*
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

As expression (*foo as bar*)

ExprSafeAsVariant

ExprSafeAsVariant fields are

annotation	smart_ptr< *rtti::TypeAnnotation* >
value	smart_ptr< *ast::Expression* >
at	*rtti::LineInfo*
fieldIndex	int
fieldFlags	*ExprFieldFieldFlags*
field	*ast::FieldDeclaration* const? const
skipQQ	bool
derefFlags	*ExprFieldDerefFlags*
printFlags	*ExprPrintFlags*
name	*builtin::das_string*
atField	*rtti::LineInfo*
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Safe as expression (*foo? as bar*)

ExprOp1

ExprOp1 fields are

atEnclosure	*rtti::LineInfo*
func	*ast::Function* ?
arguments	vector<smart_ptr<Expression>>
at	*rtti::LineInfo*
op	*builtin::das_string*
printFlags	*ExprPrintFlags*
stackTop	uint
name	*builtin::das_string*
subexpr	smart_ptr< *ast::Expression* >
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Single operator expression (+*a* or -*a* or !*a* or ~*a*)

ExprReturn

ExprReturn fields are

at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
stackTop	uint
subexpr	smart_ptr< *ast::Expression* >
block	*ast::ExprBlock* ?
genFlags	*ExprGenFlags*
refStackTop	uint
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
returnFlags	*ExprReturnFlags*
__rtti	string const

Return expression (*return* or *return foo*, or *return <- foo*)

ExprYield

ExprYield fields are

at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
subexpr	smart_ptr< *ast::Expression* >
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*
returnFlags	*ExprYieldFlags*

Yield expression (*yield foo* or *yeild <- bar*)

ExprBreak

ExprBreak fields are

at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Break expression (*break*)

ExprContinue

ExprContinue fields are

at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Continue expression (*continue*)

ExprConst

ExprConst fields are

at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Compilation time constant expression base class

ExprFakeContext

ExprFakeContext fields are

at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Compilation time only fake context expression. Will simulate as current evaluation *Context*.

ExprFakeLineInfo

ExprFakeLineInfo fields are

at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Compilation time only fake lineinfo expression. Will simulate as current file and line *LineInfo*.

ExprConstPtr

ExprConstPtr fields are

value	void?
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Null (*null*). Technically can be any other pointer, but it is used for nullptr.

ExprConstInt8

ExprConstInt8 fields are

value	int8
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds int8 constant.

ExprConstInt16

ExprConstInt16 fields are

value	int16
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds int16 constant.

ExprConstInt64

ExprConstInt64 fields are

value	int64
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds int64 constant.

ExprConstInt

ExprConstInt fields are

value	int
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds int constant.

ExprConstInt2

ExprConstInt2 fields are

value	int2
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds int2 constant.

ExprConstInt3

ExprConstInt3 fields are

value	int3
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds int3 constant.

ExprConstInt4

ExprConstInt4 fields are

value	int4
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds int4 constant.

ExprConstUInt8

ExprConstUInt8 fields are

value	uint8
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds uint8 constant.

ExprConstUInt16

ExprConstUInt16 fields are

value	uint16
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds uint16 constant.

ExprConstUInt64

ExprConstUInt64 fields are

value	uint64
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds uint64 constant.

ExprConstUInt

ExprConstUInt fields are

value	uint
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds uint constant.

ExprConstUInt2

ExprConstUInt2 fields are

value	uint2
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds uint2 constant.

ExprConstUInt3

ExprConstUInt3 fields are

value	uint3
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds uint3 constant.

ExprConstUInt4

ExprConstUInt4 fields are

value	uint4
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds uint4 constant.

ExprConstRange

ExprConstRange fields are

value	range
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds range constant.

ExprConstURange

ExprConstURange fields are

value	urange
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds urange constant.

ExprConstRange64

ExprConstRange64 fields are

value	range64
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds range64 constant.

ExprConstURange64

ExprConstURange64 fields are

value	urange64
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds urange64 constant.

ExprConstFloat

ExprConstFloat fields are

value	float
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds float constant.

ExprConstFloat2

ExprConstFloat2 fields are

value	float2
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds float2 constant.

ExprConstFloat3

ExprConstFloat3 fields are

value	float3
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds float3 constant.

ExprConstFloat4

ExprConstFloat4 fields are

value	float4
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds float4 constant.

ExprConstDouble

ExprConstDouble fields are

value	double
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds double constant.

ExprConstBool

ExprConstBool fields are

value	bool
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds bool constant.

CaptureEntry

CaptureEntry fields are

name	*builtin::das_string*
mode	*ast::CaptureMode*

Single entry in lambda capture.

ExprMakeBlock

ExprMakeBlock fields are

mmFlags	*ExprMakeBlockFlags*
at	*rtti::LineInfo*
capture	vector<CaptureEntry>
printFlags	*ExprPrintFlags*
stackTop	uint
genFlags	*ExprGenFlags*
_block	smart_ptr< *ast::Expression* >
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Any closure. Holds block as well as capture information in *CaptureEntry*.

ExprMakeGenerator

ExprMakeGenerator fields are

atEnclosure	*rtti::LineInfo*
arguments	vector<smart_ptr<Expression>>
at	*rtti::LineInfo*
capture	vector<CaptureEntry>
printFlags	*ExprPrintFlags*
name	*builtin::das_string*
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
iterType	smart_ptr< *ast::TypeDecl* >
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Generator closure (*generator<int>* or *generator<Foo&>*)

ExprMemZero

ExprMemZero fields are

atEnclosure	*rtti::LineInfo*
arguments	vector<smart_ptr<Expression>>
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
name	*builtin::das_string*
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Memzero (*memzero(expr)*)

ExprConstEnumeration

ExprConstEnumeration fields are

value	*builtin::das_string*
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
enumType	smart_ptr< *ast::Enumeration* >
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Holds enumeration constant, both type and entry (*Foo bar*).

ExprConstBitfield

ExprConstBitfield fields are

value	bitfield<>
at	*rtti::LineInfo*
bitfieldType	smart_ptr< *ast::TypeDecl* >
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Holds bitfield constant (*Foo bar*).

ExprConstString

ExprConstString fields are

value	*builtin::das_string*
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
baseType	*rtti::Type*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Holds string constant.

ExprUnsafe

ExprUnsafe fields are

at	*rtti::LineInfo*
body	smart_ptr< *ast::Expression* >
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Unsafe expression (*unsafe(addr(x))*)

VisitorAdapter

Adapter for the *AstVisitor* interface.

FunctionAnnotation

Adapter for the *AstFunctionAnnotation*.

StructureAnnotation

Adapter for the *AstStructureAnnotation*.

EnumerationAnnotation

Adapater for the *AstEnumearationAnnotation*.

PassMacro

PassMacro fields are

name	*builtin::das_string*

Adapter for the *AstPassMacro*.

ReaderMacro

ReaderMacro fields are

_module	*rtti::Module* ?
name	*builtin::das_string*

Adapter for the *AstReaderMacro*.

CommentReader

Adapter for the *AstCommentReader*.

CallMacro

CallMacro fields are

_module	*rtti::Module* ?
name	*builtin::das_string*

Adapter for the *AstCallMacro*.

VariantMacro

VariantMacro fields are

name	*builtin::das_string*

Adapter for the *AstVariantMacro*.

ForLoopMacro

ForLoopMacro fields are

name	*builtin::das_string*

Adapter for the 'AstForLoopMacro'.

CaptureMacro

CaptureMacro fields are

name	*builtin::das_string*

Adapter for the *AstCaptureMacro*.

SimulateMacro

SimulateMacro fields are

name	*builtin::das_string*

Adapter for the *AstSimulateMacro*.

ExprReader

ExprReader fields are

macro	smart_ptr< *ast::ReaderMacro* >
sequence	*builtin::das_string*
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
__rtti	string const
flags	*ExprFlags*

Compilation time only expression which holds temporary information for the *AstReaderMacro*.

ExprCallMacro

ExprCallMacro fields are

atEnclosure	*rtti::LineInfo*
arguments	vector<smart_ptr<Expression>>
macro	*ast::CallMacro* ?
at	*rtti::LineInfo*
printFlags	*ExprPrintFlags*
name	*builtin::das_string*
argumentsFailedToInfer	bool
genFlags	*ExprGenFlags*
_type	smart_ptr< *ast::TypeDecl* >
flags	*ExprFlags*
__rtti	string const

Compilation time only expression which holds temporary infromation for the *AstCallMacro*.

12.4 Call macros

quote

Returns ast expression tree of the input, without evaluating or infering it. This is useful for macros which generate code as a shortcut for generating boilerplate code.

12.5 Typeinfo macros

ast_typedecl

Returns TypeDeclPtr of the type specified via type<> or subexpression type, for example typeinfo(ast_typedecl type<int?>)

ast_function

Returns FunctionPtr to the function specified by subexrepssion, for example typeinfo(ast_function @@foo)

12.6 Handled types

MakeStruct

Part of *ExprMakeStruct*, happens to be vector of *MakeFieldDecl*.

12.7 Classes

AstFunctionAnnotation

Annotation macro which is attached to the *Function*.

it defines as follows

`AstFunctionAnnotation`.**transform**(*self: AstFunctionAnnotation; call: smart_ptr<ast::ExprCallFunc>; errors: das_string*)

transform returns *ExpressionPtr*

argument	argument type
self	*ast::AstFunctionAnnotation*
call	smart_ptr< *ast::ExprCallFunc* >
errors	*builtin::das_string*

This callback occurs during the *infer* pass of the compilation. If no transformation is needed, the callback should return *null*. *errors* is filled with the transformation errors should they occur. Returned value replaces function call in the ast.

`AstFunctionAnnotation`.**verifyCall**(*self: AstFunctionAnnotation; call: smart_ptr<ast::ExprCallFunc>; args: AnnotationArgumentList const; progArgs: AnnotationArgumentList const; errors: das_string*) verifyCall returns bool

argument	argument type
self	*ast::AstFunctionAnnotation*
call	smart_ptr< *ast::ExprCallFunc* >
args	*rtti::AnnotationArgumentList* const
progArgs	*rtti::AnnotationArgumentList* const
errors	*builtin::das_string*

This callback occurs during the *lint* pass of the compilation. If call has lint errors it should return *false* and *errors* is filled with the lint errors.

`AstFunctionAnnotation.`**`apply`**(*self: AstFunctionAnnotation; func: FunctionPtr; group: ModuleGroup; args: AnnotationArgumentList const; errors: das_string*)

apply returns bool

argument	argument type
self	*ast::AstFunctionAnnotation*
func	*FunctionPtr*
group	*rtti::ModuleGroup*
args	*rtti::AnnotationArgumentList* const
errors	*builtin::das_string*

This callback occurs during the *parse* pass of the compilation on the function itself. If function has application errors it should return *false* and *errors* field.

`AstFunctionAnnotation.`**`generic_apply`**(*self: AstFunctionAnnotation; func: FunctionPtr; group: ModuleGroup; args: AnnotationArgumentList const; errors: das_string*)

generic_apply returns bool

argument	argument type
self	*ast::AstFunctionAnnotation*
func	*FunctionPtr*
group	*rtti::ModuleGroup*
args	*rtti::AnnotationArgumentList* const
errors	*builtin::das_string*

This call occurs during the *infer* pass of the compilation, when generic function is instanced on the instance of the function. If function has application errors it should return *false* and *errors* field.

AstFunctionAnnotation.**finish**(*self: AstFunctionAnnotation; func: FunctionPtr; group: ModuleGroup; args: AnnotationArgumentList const; progArgs: AnnotationArgumentList const; errors: das_string*)

finish returns bool

argument	argument type
self	*ast::AstFunctionAnnotation*
func	*FunctionPtr*
group	*rtti::ModuleGroup*
args	*rtti::AnnotationArgumentList* const
progArgs	*rtti::AnnotationArgumentList* const
errors	*builtin::das_string*

This callback occurs during the *finalize allocations* pass of the compilation, after the stack is allocated, on the function itself. If function has finalization errors it should return *false* and *errors* field.

AstFunctionAnnotation.**patch**(*self: AstFunctionAnnotation; func: FunctionPtr; group: ModuleGroup; args: AnnotationArgumentList const; progArgs: AnnotationArgumentList const; errors: das_string; astChanged: bool&*)

patch returns bool

argument	argument type
self	*ast::AstFunctionAnnotation*
func	*FunctionPtr*
group	*rtti::ModuleGroup*
args	*rtti::AnnotationArgumentList* const
progArgs	*rtti::AnnotationArgumentList* const
errors	*builtin::das_string*
astChanged	bool&

This callback occurs right after the *infer* pass of the compilation on the function itself. If function has patching errors it should return *false* and *errors* field. If the *astChanged* flag is set, *infer* pass will be repeated. This allows to fix up the function after the *infer* pass with all the type information fully available.

`AstFunctionAnnotation.`**`fixup`**`(`*self: AstFunctionAnnotation; func: FunctionPtr; group: ModuleGroup; args: AnnotationArgumentList const; progArgs: AnnotationArgumentList const; errors: das_string*`)`

fixup returns bool

argument	argument type
self	*ast::AstFunctionAnnotation*
func	*FunctionPtr*
group	*rtti::ModuleGroup*
args	*rtti::AnnotationArgumentList* const
progArgs	*rtti::AnnotationArgumentList* const
errors	*builtin::das_string*

This callback occurs during the *finalize allocations* pass of the compilation, before the stack is allocated, on the function itself. If function has fixup errors it should return *false* and *errors* field.

`AstFunctionAnnotation.`**`lint`**`(`*self: AstFunctionAnnotation; func: FunctionPtr; group: ModuleGroup; args: AnnotationArgumentList const; progArgs: AnnotationArgumentList const; errors: das_string*`)`

lint returns bool

argument	argument type
self	*ast::AstFunctionAnnotation*
func	*FunctionPtr*
group	*rtti::ModuleGroup*
args	*rtti::AnnotationArgumentList* const
progArgs	*rtti::AnnotationArgumentList* const
errors	*builtin::das_string*

This callback occurs during the *lint* pass of the compilation on the function itself. If function has lint errors it should return *false* and *errors* field.

`AstFunctionAnnotation.`**`complete`**(*self: AstFunctionAnnotation; func: FunctionPtr; ctx: smart_ptr<rtti::Context>*)

argument	argument type
self	*ast::AstFunctionAnnotation*
func	*FunctionPtr*
ctx	smart_ptr< *rtti::Context* >

This callback occurs as the final stage of *Context* simulation.

`AstFunctionAnnotation.`**`isCompatible`**(*self: AstFunctionAnnotation; func: FunctionPtr; types: VectorTypeDeclPtr; decl: AnnotationDeclaration const; errors: das_string*)

isCompatible returns bool

argument	argument type
self	*ast::AstFunctionAnnotation*
func	*FunctionPtr*
types	*VectorTypeDeclPtr*
decl	*rtti::AnnotationDeclaration* const
errors	*builtin::das_string*

This callback occurs during function type matching for both generic and regular functions. If function can accept given argument types it should return *true*, otherwise *errors* is filled with the matching problems.

`AstFunctionAnnotation.`**`isSpecialized`**(*self: AstFunctionAnnotation*) isSpecialized returns bool

This callback occurs during function type matching. If function requires special type matching (i.e. *isCompatible* ` is implemented) it should return *true*.

`AstFunctionAnnotation.`**`appendToMangledName`**(*self: AstFunctionAnnotation; func: FunctionPtr; decl: AnnotationDeclaration const; mangledName: das_string*)

argument	argument type
self	*ast::AstFunctionAnnotation*
func	*FunctionPtr*
decl	*rtti::AnnotationDeclaration* const
mangledName	*builtin::das_string*

This call occurs when the function mangled name is requested. This is the way for the macro to ensure function is unique, even though type signature may be identical.

`AstBlockAnnotation`

Annotation macro which is attached to the *ExprBlock*.

it defines as follows

`AstBlockAnnotation.`**`apply`**(*self: AstBlockAnnotation; blk: smart_ptr<ast::ExprBlock>; group: ModuleGroup; args: AnnotationArgumentList const; errors: das_string*)

apply returns bool

argument	argument type
self	*ast::AstBlockAnnotation*
blk	smart_ptr< *ast::ExprBlock* >
group	*rtti::ModuleGroup*
args	*rtti::AnnotationArgumentList* const
errors	*builtin::das_string*

This callback occurs during the *parse* pass of the compilation. If block has application errors it should return *false* and *errors* field.

`AstBlockAnnotation.`**`finish`**(*self: AstBlockAnnotation; blk: smart_ptr<ast::ExprBlock>; group: ModuleGroup; args: AnnotationArgumentList const; progArgs: AnnotationArgumentList const; errors: das_string*)

finish returns bool

argument	argument type
self	*ast::AstBlockAnnotation*
blk	smart_ptr< *ast::ExprBlock* >
group	*rtti::ModuleGroup*
args	*rtti::AnnotationArgumentList* const
progArgs	*rtti::AnnotationArgumentList* const
errors	*builtin::das_string*

This callback occurs during the *finalize allocations* pass of the compilation, after the stack is allocated. If block has finalization errors it should return *false* and *errors* field.

AstStructureAnnotation

Annotation macro which is attached to the *Structure*.

it defines as follows

`AstStructureAnnotation.`**apply**(*self: AstStructureAnnotation; st: StructurePtr; group: ModuleGroup; args: AnnotationArgumentList const; errors: das_string*)

apply returns bool

argument	argument type
self	*ast::AstStructureAnnotation*
st	*StructurePtr*
group	*rtti::ModuleGroup*
args	*rtti::AnnotationArgumentList* const
errors	*builtin::das_string*

This callback occurs during the *parse* pass of the compilation. If structure has application errors it should return *false* and *errors* field.

`AstStructureAnnotation.`**finish**(*self: AstStructureAnnotation; st: StructurePtr; group: ModuleGroup; args: AnnotationArgumentList const; errors: das_string*)

finish returns bool

argument	argument type
self	*ast::AstStructureAnnotation*
st	*StructurePtr*
group	*rtti::ModuleGroup*
args	*rtti::AnnotationArgumentList* const
errors	*builtin::das_string*

This callback occurs during the *finalize allocations* pass of the compilation, after the stack is allocated. If structure has finalization errors it should return *false* and *errors* field.

`AstStructureAnnotation.`**`patch`**(*self: AstStructureAnnotation; st: StructurePtr; group: ModuleGroup; args: AnnotationArgumentList const; errors: das_string; astChanged: bool&*)

patch returns bool

argument	argument type
self	*ast::AstStructureAnnotation*
st	*StructurePtr*
group	*rtti::ModuleGroup*
args	*rtti::AnnotationArgumentList* const
errors	*builtin::das_string*
astChanged	bool&

This callback occurs right after the *infer* pass of the compilation on the structure itself. If structure has patching errors it should return *false* and *errors* field. If the *astChanged* flag is set, *infer* pass will be repeated. This allows to fix up the function after the *infer* pass with all the type information fully available.

`AstStructureAnnotation.`**`complete`**(*self: AstStructureAnnotation; st: StructurePtr; ctx: smart_ptr<rtti::Context>*)

argument	argument type
self	*ast::AstStructureAnnotation*
st	*StructurePtr*
ctx	smart_ptr< *rtti::Context* >

This callback occurs as the final stage of *Context* simulation.

AstStructureAnnotation.**aotPrefix**(*self: AstStructureAnnotation; st: StructurePtr; args: AnnotationArgumentList const; writer: StringBuilderWriter*)

argument	argument type
self	*ast::AstStructureAnnotation*
st	*StructurePtr*
args	*rtti::AnnotationArgumentList* const
writer	*strings::StringBuilderWriter*

This callback occurs during the *AOT*. It is used to generate CPP code before the structure declaration.

AstStructureAnnotation.**aotBody**(*self: AstStructureAnnotation; st: StructurePtr; args: AnnotationArgumentList const; writer: StringBuilderWriter*)

argument	argument type
self	*ast::AstStructureAnnotation*
st	*StructurePtr*
args	*rtti::AnnotationArgumentList* const
writer	*strings::StringBuilderWriter*

This callback occurs during the *AOT*. It is used to generate CPP code in the body of the structure.

AstStructureAnnotation.**aotSuffix**(*self: AstStructureAnnotation; st: StructurePtr; args: AnnotationArgumentList const; writer: StringBuilderWriter*)

argument	argument type
self	*ast::AstStructureAnnotation*
st	*StructurePtr*
args	*rtti::AnnotationArgumentList* const
writer	*strings::StringBuilderWriter*

This callback occurs during the *AOT*. It is used to generate CPP code after the structure declaration.

AstPassMacro

This macro is used to implement custom *infer* passes.

it defines as follows

AstPassMacro.**apply**(*self: AstPassMacro; prog: ProgramPtr; mod: rtti::Module? const*)

apply returns bool

argument	argument type
self	*ast::AstPassMacro*
prog	*ProgramPtr*
mod	*rtti::Module* ? const

This callback is called after *infer* pass. If macro did any work it returns *true*; *infer* pass is restarted a the memoent when first macro which did any work.

AstVariantMacro

This macro is used to implement custom *is*, *as* and *?as* expressions.

it defines as follows

`AstVariantMacro.`**`visitExprIsVariant`** (*self: AstVariantMacro; prog: ProgramPtr; mod: rtti::Module? const; expr: smart_ptr<ast::ExprIsVariant> const*)

visitExprIsVariant returns *ExpressionPtr*

argument	argument type
self	*ast::AstVariantMacro*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
expr	smart_ptr< *ast::ExprIsVariant* > const

This callback occurs during the *infer* pass for every *ExprIsVariant* (a *is* b). If no work is necessary it should return *null*, otherwise expression will be replaced by the result.

`AstVariantMacro.`**`visitExprAsVariant`** (*self: AstVariantMacro; prog: ProgramPtr; mod: rtti::Module? const; expr: smart_ptr<ast::ExprAsVariant> const*)

visitExprAsVariant returns *ExpressionPtr*

argument	argument type
self	*ast::AstVariantMacro*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
expr	smart_ptr< *ast::ExprAsVariant* > const

This callback occurs during the *infer* pass for every *ExprAsVariant* (a *as* b). If no work is necessary it should return *null*, otherwise expression will be replaced by the result.

`AstVariantMacro.`**`visitExprSafeAsVariant`**(*self: AstVariantMacro; prog: ProgramPtr; mod: rtti::Module? const; expr: smart_ptr<ast::ExprSafeAsVariant> const*)

visitExprSafeAsVariant returns *ExpressionPtr*

argument	argument type
self	*ast::AstVariantMacro*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
expr	smart_ptr< *ast::ExprSafeAsVariant* > const

This callback occurs during the *infer* pass for every *ExprSafeIsVariant* (a *?as* b). If no work is necessary it should return *null*, otherwise expression will be replaced by the result.

`AstForLoopMacro`

This macro is used to implement custom for-loop handlers. It is similar to visitExprFor callback of the AstVisitor.

it defines as follows

`AstForLoopMacro.`**`visitExprFor`**(*self: AstForLoopMacro; prog: ProgramPtr; mod: rtti::Module? const; expr: smart_ptr<ast::ExprFor> const*)

visitExprFor returns *ExpressionPtr*

argument	argument type
self	*ast::AstForLoopMacro*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
expr	smart_ptr< *ast::ExprFor* > const

This callback occurs during the *infer* pass for every *ExprFor*. If no work is necessary it should return *null*, otherwise expression will be replaced by the result.

`AstCaptureMacro`

This macro is used to implement custom lambda capturing functionality.

it defines as follows

`AstCaptureMacro.`**`captureExpression`**(*self: AstCaptureMacro; prog: rtti::Program? const; mod: rtti::Module? const; expr: ExpressionPtr; etype: TypeDeclPtr*) captureExpression returns *ExpressionPtr*

argument	argument type
self	*ast::AstCaptureMacro*
prog	*rtti::Program* ? const
mod	*rtti::Module* ? const
expr	*ExpressionPtr*
etype	*TypeDeclPtr*

This callback occurs during the 'infer' pass for every time a lambda expression (or generator) is captured for every captured expression.

`AstCaptureMacro.`**`captureFunction`**(*self: AstCaptureMacro; prog: rtti::Program? const; mod: rtti::Module? const; lcs: ast::Structure?; fun: FunctionPtr*)

argument	argument type
self	*ast::AstCaptureMacro*
prog	*rtti::Program* ? const
mod	*rtti::Module* ? const
lcs	*ast::Structure* ?
fun	*FunctionPtr*

This callback occurs during the 'infer' pass for every time a lambda expression (or generator) is captured, for every generated lambda (or generator) function.

`AstSimulateMacro`

Macro which is attached to the context simulation.

it defines as follows

`AstSimulateMacro.`**`preSimulate`**(*self: AstSimulateMacro; prog: rtti::Program? const; ctx: rtti::Context? const*)

preSimulate returns bool

argument	argument type
self	*ast::AstSimulateMacro*
prog	*rtti::Program* ? const
ctx	*rtti::Context* ? const

This callback occurs before the context simulation.

`AstSimulateMacro.`**`simulate`** (*self: AstSimulateMacro; prog: rtti::Program? const; ctx: rtti::Context? const*)

simulate returns bool

argument	argument type
self	*ast::AstSimulateMacro*
prog	*rtti::Program* ? const
ctx	*rtti::Context* ? const

This callback occurs after the context simulation.

`AstReaderMacro`

This macro is used to implement custom parsing functionality, i.e. anything starting with %NameOfTheMacro~ and ending when the macro says it ends.

it defines as follows

`AstReaderMacro.`**`accept`** (*self: AstReaderMacro; prog: ProgramPtr; mod: rtti::Module? const; expr: ast::ExprReader? const; ch: int const; info: LineInfo const*)

accept returns bool

argument	argument type
self	*ast::AstReaderMacro*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
expr	*ast::ExprReader* ? const
ch	int const
info	*rtti::LineInfo* const

This callback occurs during the *parse* pass for every character. When the macro is done with the input (i.e. recognizeable input ends) it should return *false*. Typically characters are appended to the *expr.sequence* inside the ExprReader.

`AstReaderMacro.`**`visit`** (*self: AstReaderMacro; prog: ProgramPtr; mod: rtti::Module? const; expr: smart_ptr<ast::ExprReader> const*)

visit returns *ExpressionPtr*

argument	argument type
self	*ast::AstReaderMacro*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
expr	smart_ptr< *ast::ExprReader* > const

This callback occurs during the *infer* pass for every instance of *ExprReader* for that specific macro. Macro needs to convert *ExprReader* to some meaningful expression.

AstCommentReader

This macro is used to implement custom comment parsing function (such as doxygen-style documentation etc).

it defines as follows

`AstCommentReader.`**`open`** (*self: AstCommentReader; prog: ProgramPtr; mod: rtti::Module? const; cpp: bool const; info: LineInfo const*)

argument	argument type
self	*ast::AstCommentReader*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
cpp	bool const
info	*rtti::LineInfo* const

This callback occurs during the *parse* pass for every // or /* sequence which indicated begining of the comment section.

`AstCommentReader.`**`accept`** (*self: AstCommentReader; prog: ProgramPtr; mod: rtti::Module? const; ch: int const; info: LineInfo const*)

argument	argument type
self	*ast::AstCommentReader*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
ch	int const
info	*rtti::LineInfo* const

This callback occurs during the *parse* pass for every character in the comment section.

AstCommentReader.**close** (*self: AstCommentReader; prog: ProgramPtr; mod: rtti::Module? const; info: LineInfo const*)

argument	argument type
self	*ast::AstCommentReader*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
info	*rtti::LineInfo* const

This callback occurs during the *parse* pass for every new line or */ sequence which indicates end of the comment section.

AstCommentReader.**beforeStructure** (*self: AstCommentReader; prog: ProgramPtr; mod: rtti::Module? const; info: LineInfo const*)

argument	argument type
self	*ast::AstCommentReader*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
info	*rtti::LineInfo* const

This callback occurs during the *parse* pass before the structure body block.

AstCommentReader.**afterStructure** (*self: AstCommentReader; st: StructurePtr; prog: ProgramPtr; mod: rtti::Module? const; info: LineInfo const*)

argument	argument type
self	*ast::AstCommentReader*
st	*StructurePtr*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
info	*rtti::LineInfo* const

This callback occurs during the *parse* pass after the structure body block.

AstCommentReader.**beforeStructureFields** (*self: AstCommentReader; prog: ProgramPtr; mod: rtti::Module? const; info: LineInfo const*)

argument	argument type
self	*ast::AstCommentReader*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
info	*rtti::LineInfo* const

This callback occurs during the *parse* pass before the first structure field is declared.

`AstCommentReader.`**`afterStructureField`**(*self: AstCommentReader; name: string const; prog: ProgramPtr; mod: rtti::Module? const; info: LineInfo const*)

argument	argument type
self	*ast::AstCommentReader*
name	string const
prog	*ProgramPtr*
mod	*rtti::Module* ? const
info	*rtti::LineInfo* const

This callback occurs during the *parse* pass after the structure field is declared (after the following comment section, should it have one).

`AstCommentReader.`**`afterStructureFields`**(*self: AstCommentReader; prog: ProgramPtr; mod: rtti::Module? const; info: LineInfo const*)

argument	argument type
self	*ast::AstCommentReader*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
info	*rtti::LineInfo* const

This callback occurs during the *parse* pass after the last structure field is declared.

`AstCommentReader.`**`beforeFunction`**(*self: AstCommentReader; prog: ProgramPtr; mod: rtti::Module? const; info: LineInfo const*)

argument	argument type
self	*ast::AstCommentReader*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
info	*rtti::LineInfo* const

This callback occurs during the *parse* pass before the function body block.

`AstCommentReader.`**`afterFunction`**(*self: AstCommentReader; fn: FunctionPtr; prog: ProgramPtr; mod: rtti::Module? const; info: LineInfo const*)

argument	argument type
self	*ast::AstCommentReader*
fn	*FunctionPtr*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
info	*rtti::LineInfo* const

This callback occurs during the *parse* pass after the function body block.

`AstCommentReader.`**`beforeGlobalVariables`**(*self: AstCommentReader; prog: ProgramPtr; mod: rtti::Module? const; info: LineInfo const*)

argument	argument type
self	*ast::AstCommentReader*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
info	*rtti::LineInfo* const

This callback occurs during the *parse* pass before the first global variable declaration but after *let* or *var* keyword.

`AstCommentReader.`**`afterGlobalVariable`**(*self: AstCommentReader; name: string const; prog: ProgramPtr; mod: rtti::Module? const; info: LineInfo const*)

argument	argument type
self	*ast::AstCommentReader*
name	string const
prog	*ProgramPtr*
mod	*rtti::Module* ? const
info	*rtti::LineInfo* const

This callback occurs during the *parse* pass after global variable is declaraed (after the following comment section, should it have one).

`AstCommentReader.`**`afterGlobalVariables`** (*self: AstCommentReader; prog: ProgramPtr; mod: rtti::Module? const; info: LineInfo const*)

argument	argument type
self	*ast::AstCommentReader*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
info	*rtti::LineInfo* const

This callback occurs during the *parse* pass after every global variable in the declaration is declared.

`AstCommentReader.`**`beforeVariant`** (*self: AstCommentReader; prog: ProgramPtr; mod: rtti::Module? const; info: LineInfo const*)

argument	argument type
self	*ast::AstCommentReader*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
info	*rtti::LineInfo* const

This callback occurs during the *parse* pass before the variant alias declaration.

`AstCommentReader.`**`afterVariant`** (*self: AstCommentReader; name: string const; prog: ProgramPtr; mod: rtti::Module? const; info: LineInfo const*)

argument	argument type
self	*ast::AstCommentReader*
name	string const
prog	*ProgramPtr*
mod	*rtti::Module* ? const
info	*rtti::LineInfo* const

This callback occurs during the *parse* after the variant alias declaration.

`AstCommentReader.`**`beforeEnumeration`**(*self: AstCommentReader; prog: ProgramPtr; mod: rtti::Module? const; info: LineInfo const*)

argument	argument type
self	*ast::AstCommentReader*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
info	*rtti::LineInfo* const

This callback occurs during the *parse* before the enumeration declaration.

`AstCommentReader.`**`afterEnumeration`**(*self: AstCommentReader; name: string const; prog: ProgramPtr; mod: rtti::Module? const; info: LineInfo const*)

argument	argument type
self	*ast::AstCommentReader*
name	string const
prog	*ProgramPtr*
mod	*rtti::Module* ? const
info	*rtti::LineInfo* const

This callback occurs during the *parse* after the enumeration declaration.

`AstCommentReader.`**`beforeAlias`**(*self: AstCommentReader; prog: ProgramPtr; mod: rtti::Module? const; info: LineInfo const*)

argument	argument type
self	*ast::AstCommentReader*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
info	*rtti::LineInfo* const

This callback occurs during the *parse* pass before the type alias declaration.

`AstCommentReader.`**`afterAlias`**(*self: AstCommentReader; name: string const; prog: ProgramPtr; mod: rtti::Module? const; info: LineInfo const*)

argument	argument type
self	*ast::AstCommentReader*
name	string const
prog	*ProgramPtr*
mod	*rtti::Module* ? const
info	*rtti::LineInfo* const

This callback occurs during the *parse* pass after the type alias declaration.

`AstCallMacro`

This macro is used to implement custom call-like expressions (like *foo(bar,bar2,. . .)*).

it defines as follows

`AstCallMacro.`**`preVisit`**(*self: AstCallMacro; prog: ProgramPtr; mod: rtti::Module? const; expr: smart_ptr<ast::ExprCallMacro> const*)

argument	argument type
self	*ast::AstCallMacro*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
expr	smart_ptr< *ast::ExprCallMacro* > const

This callback occurs during the *infer* pass for every *ExprCallMacro*, before its arguments are inferred.

`AstCallMacro.`**`visit`**(*self: AstCallMacro; prog: ProgramPtr; mod: rtti::Module? const; expr: smart_ptr<ast::ExprCallMacro> const*) visit returns *ExpressionPtr*

argument	argument type
self	*ast::AstCallMacro*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
expr	smart_ptr< *ast::ExprCallMacro* > const

This callback occurs during the *infer* pass for every *ExprCallMacro*, after its arguments are inferred. When fully inferred macro is expected to replace *ExprCallMacro* with meaningful expression.

`AstCallMacro.`**`canVisitArgument`** (*self: AstCallMacro; expr: smart_ptr<ast::ExprCallMacro> const; argIndex: int const*)

canVisitArgument returns bool

argument	argument type
self	*ast::AstCallMacro*
expr	smart_ptr< *ast::ExprCallMacro* > const
argIndex	int const

This callback occurs during the *infer* pass before the arguments of the call macro are visited. If callback returns true, the argument of given index is visited, otherwise it acts like a query expression.

`AstCallMacro.`**`canFoldReturnResult`** (*self: AstCallMacro; expr: smart_ptr<ast::ExprCallMacro> const*)

canFoldReturnResult returns bool

argument	argument type
self	*ast::AstCallMacro*
expr	smart_ptr< *ast::ExprCallMacro* > const

If true the enclosing function can infer return result as *void* when unspecified. If false function will have to wait for the macro to fold.

`AstTypeInfoMacro`

This macro is used to implement type info traits, i.e. *typeinfo(YourTraitHere . . .)* expressions.

it defines as follows

`AstTypeInfoMacro.`**`getAstChange`** (*self: AstTypeInfoMacro; expr: smart_ptr<ast::ExprTypeInfo> const; errors: das_string*) getAstChange returns *ExpressionPtr*

argument	argument type
self	*ast::AstTypeInfoMacro*
expr	smart_ptr< *ast::ExprTypeInfo* > const
errors	*builtin::das_string*

This callback occurs during the *infer* pass. If no changes are necessary it should return *null*, otherwise expression will be replaced by the result. *errors* should be filled if trait is malformed.

`AstTypeInfoMacro.`**`getAstType`**(*self: AstTypeInfoMacro; lib: ModuleLibrary; expr: smart_ptr<ast::ExprTypeInfo> const; errors: das_string*)

getAstType returns *TypeDeclPtr*

argument	argument type
self	*ast::AstTypeInfoMacro*
lib	*ast::ModuleLibrary*
expr	smart_ptr< *ast::ExprTypeInfo* > const
errors	*builtin::das_string*

This callback occurs during the *infer* pass. It should return type of the typeinfo expression. That way trait can return *Type*, and not *Expression*.

`AstEnumerationAnnotation`

Annotation macro which is attached to *Enumeration*.

it defines as follows

`AstEnumerationAnnotation.`**`apply`**(*self: AstEnumerationAnnotation; st: EnumerationPtr; group: ModuleGroup; args: AnnotationArgumentList const; errors: das_string*)

apply returns bool

argument	argument type
self	*ast::AstEnumerationAnnotation*
st	*EnumerationPtr*
group	*rtti::ModuleGroup*
args	*rtti::AnnotationArgumentList* const
errors	*builtin::das_string*

This callback occurs during the *parse* pass. If any errors occur *errors* should be filled and *false* should be returned.

AstVisitor

This class implements *Visitor* interface for the ast tree. For typical expression two methods are provided: *preVisitExpr* and *visitExpr*. *preVisitExpr* occurs before the subexpressions are visited, and *visitExpr* occurs after the subexpressions are visited. *visitExpr* can return new expression which will replace the original one, or original expression - if no changes are necessary. There are other potential callbacks deppending of the nature of expression, which represent particular sections of the ast tree. Additionally 'preVisitExpression' and *visitExpression* are called before and after expression specific callbacks.

it defines as follows

AstVisitor.**preVisitProgram**(*self: AstVisitor; prog: ProgramPtr*)

argument	argument type
self	*ast::AstVisitor*
prog	*ProgramPtr*

before entire program, put your initialization there.

AstVisitor.**visitProgram**(*self: AstVisitor; porg: ProgramPtr*)

argument	argument type
self	*ast::AstVisitor*
porg	*ProgramPtr*

after entire program, put your finalizers there.

AstVisitor.**preVisitProgramBody**(*self: AstVisitor; prog: ProgramPtr; mod: rtti::Module? const*)

argument	argument type
self	*ast::AstVisitor*
prog	*ProgramPtr*
mod	*rtti::Module* ? const

after enumerations, structures, and aliases, but before global variables, generics and functions.

`AstVisitor.`**`preVisitModule`**(*self: AstVisitor; mod: rtti::Module? const*)

argument	argument type
self	*ast::AstVisitor*
mod	*rtti::Module* ? const

before each module

`AstVisitor.`**`visitModule`**(*self: AstVisitor; mod: rtti::Module? const*)

argument	argument type
self	*ast::AstVisitor*
mod	*rtti::Module* ? const

after each module

`AstVisitor.`**`preVisitExprTypeDecl`**(*self: AstVisitor; expr: smart_ptr<ast::ExprTypeDecl> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprTypeDecl* > const

before *ExprTypeDecl*

`AstVisitor.`**`visitExprTypeDecl`**(*self: AstVisitor; expr: smart_ptr<ast::ExprTypeDecl> const*)

visitExprTypeDecl returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprTypeDecl* > const

after *ExprTypeDecl*

`AstVisitor.`**`preVisitTypeDecl`**(*self: AstVisitor; typ: TypeDeclPtr*)

argument	argument type
self	*ast::AstVisitor*
typ	*TypeDeclPtr*

before a type declaration anywhere. yor type validation code typically goes here

`AstVisitor.`**`visitTypeDecl`**(*self: AstVisitor; typ: TypeDeclPtr*)

visitTypeDecl returns *TypeDeclPtr*

argument	argument type
self	*ast::AstVisitor*
typ	*TypeDeclPtr*

after a type declaration

`AstVisitor.`**`preVisitAlias`**(*self: AstVisitor; typ: TypeDeclPtr; name: das_string const*)

argument	argument type
self	*ast::AstVisitor*
typ	*TypeDeclPtr*
name	*builtin::das_string* const

before *TypeDecl*

`AstVisitor.`**`visitAlias`**(*self: AstVisitor; typ: TypeDeclPtr; name: das_string const*)

visitAlias returns *TypeDeclPtr*

argument	argument type
self	*ast::AstVisitor*
typ	*TypeDeclPtr*
name	*builtin::das_string* const

after *TypeDecl*

`AstVisitor.`**`canVisitEnumeration`**(*self: AstVisitor; arg: ast::Enumeration? const*) canVisitEnumeration returns bool

argument	argument type
self	*ast::AstVisitor*
arg	*ast::Enumeration* ? const

if true *Enumeration* will be visited

`AstVisitor.`**`preVisitEnumeration`**(*self: AstVisitor; enu: EnumerationPtr*)

argument	argument type
self	*ast::AstVisitor*
enu	*EnumerationPtr*

before *Enumeration*

`AstVisitor.`**`preVisitEnumerationValue`**(*self: AstVisitor; enu: EnumerationPtr; name: das_string const; value: ExpressionPtr; last: bool const*)

argument	argument type
self	*ast::AstVisitor*
enu	*EnumerationPtr*
name	*builtin::das_string* const
value	*ExpressionPtr*
last	bool const

before every enumeration entry

`AstVisitor.`**`visitEnumerationValue`**(*self: AstVisitor; enu: EnumerationPtr; name: das_string const; value: ExpressionPtr; last: bool const*)

visitEnumerationValue returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
enu	*EnumerationPtr*
name	*builtin::das_string* const
value	*ExpressionPtr*
last	bool const

after every enumeration entry

`AstVisitor.`**`visitEnumeration`**(*self: AstVisitor; enu: EnumerationPtr*)

visitEnumeration returns *EnumerationPtr*

argument	argument type
self	*ast::AstVisitor*
enu	*EnumerationPtr*

after *Enumeration*

`AstVisitor.`**`canVisitStructure`**(*self: AstVisitor; arg: ast::Structure? const*)

canVisitStructure returns bool

argument	argument type
self	*ast::AstVisitor*
arg	*ast::Structure* ? const

if true *Structure* will be visited

`AstVisitor.`**`preVisitStructure`**(*self: AstVisitor; str: StructurePtr*)

argument	argument type
self	*ast::AstVisitor*
str	*StructurePtr*

before *Structure*

`AstVisitor.`**`preVisitStructureField`**(*self: AstVisitor; str: StructurePtr; decl: FieldDeclaration const; last: bool const*)

argument	argument type
self	*ast::AstVisitor*
str	*StructurePtr*
decl	*ast::FieldDeclaration* const
last	bool const

before every structure field

`AstVisitor.`**`visitStructureField`**(*self: AstVisitor; str: StructurePtr; decl: FieldDeclaration const; last: bool const*)

argument	argument type
self	*ast::AstVisitor*
str	*StructurePtr*
decl	*ast::FieldDeclaration* const
last	bool const

after every structure field

`AstVisitor.`**`visitStructure`**(*self: AstVisitor; str: StructurePtr*)

visitStructure returns *StructurePtr*

argument	argument type
self	*ast::AstVisitor*
str	*StructurePtr*

after *Structure*

`AstVisitor.`**`canVisitFunction`**(*self: AstVisitor; fun: ast::Function? const*)

canVisitFunction returns bool

argument	argument type
self	*ast::AstVisitor*
fun	*ast::Function* ? const

if true *Function* will be visited

`AstVisitor.`**`canVisitFunctionArgumentInit`**(*self: AstVisitor; fun: ast::Function? const; arg: VariablePtr; value: ExpressionPtr*)

canVisitFunctionArgumentInit returns bool

argument	argument type
self	*ast::AstVisitor*
fun	*ast::Function* ? const
arg	*VariablePtr*
value	*ExpressionPtr*

if true function argument initialization expressions will be visited

`AstVisitor.`**`preVisitFunction`**(*self: AstVisitor; fun: FunctionPtr*)

argument	argument type
self	*ast::AstVisitor*
fun	*FunctionPtr*

before *Function*

`AstVisitor.`**`visitFunction`**(*self: AstVisitor; fun: FunctionPtr*)

visitFunction returns *FunctionPtr*

argument	argument type
self	*ast::AstVisitor*
fun	*FunctionPtr*

after *Function*

`AstVisitor.`**`preVisitFunctionArgument`**(*self: AstVisitor; fun: FunctionPtr; arg: VariablePtr; lastArg: bool const*)

argument	argument type
self	*ast::AstVisitor*
fun	*FunctionPtr*
arg	*VariablePtr*
lastArg	bool const

before every argument

`AstVisitor.`**`visitFunctionArgument`**(*self: AstVisitor; fun: FunctionPtr; arg: VariablePtr; lastArg: bool const*)

visitFunctionArgument returns *VariablePtr*

argument	argument type
self	*ast::AstVisitor*
fun	*FunctionPtr*
arg	*VariablePtr*
lastArg	bool const

after every argument

`AstVisitor.`**`preVisitFunctionArgumentInit`**(*self: AstVisitor; fun: FunctionPtr; arg: VariablePtr; value: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
fun	*FunctionPtr*
arg	*VariablePtr*
value	*ExpressionPtr*

before every argument initialization expression (should it have one), between 'preVisitFunctionArgument' and *visitFunctionArgument*

`AstVisitor.`**`visitFunctionArgumentInit`**(*self: AstVisitor; fun: FunctionPtr; arg: VariablePtr; value: ExpressionPtr*)

visitFunctionArgumentInit returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
fun	*FunctionPtr*
arg	*VariablePtr*
value	*ExpressionPtr*

after every argument initialization expression (should it have one), between 'preVisitFunctionArgument' and *visitFunctionArgument*

`AstVisitor.`**`preVisitFunctionBody`**`(`*self: AstVisitor; fun: FunctionPtr; expr: ExpressionPtr*`)`

argument	argument type
self	*ast::AstVisitor*
fun	*FunctionPtr*
expr	*ExpressionPtr*

before the *Function* body block, between *preVisitFunction* and *visitFunction* (not for abstract functions)

`AstVisitor.`**`visitFunctionBody`**`(`*self: AstVisitor; fun: FunctionPtr; expr: ExpressionPtr*`)`

visitFunctionBody returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
fun	*FunctionPtr*
expr	*ExpressionPtr*

after the *Function* body block, between *preVisitFunction* and *visitFunction* (not for abstract functions)

`AstVisitor.`**`preVisitExpression`**`(`*self: AstVisitor; expr: ExpressionPtr*`)`

argument	argument type
self	*ast::AstVisitor*
expr	*ExpressionPtr*

before every *Expression*

`AstVisitor.`**`visitExpression`**`(`*self: AstVisitor; expr: ExpressionPtr*`)`

visitExpression returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	*ExpressionPtr*

after every *Expression*

`AstVisitor.`**`preVisitExprBlock`**`(`*self: AstVisitor; blk: smart_ptr<ast::ExprBlock> const*`)`

argument	argument type
self	*ast::AstVisitor*
blk	smart_ptr< *ast::ExprBlock* > const

before *ExprBlock*

`AstVisitor.`**`visitExprBlock`**(*self: AstVisitor; blk: smart_ptr<ast::ExprBlock> const*)

visitExprBlock returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
blk	smart_ptr< *ast::ExprBlock* > const

after *ExprBlock*

`AstVisitor.`**`preVisitExprBlockArgument`**(*self: AstVisitor; blk: smart_ptr<ast::ExprBlock> const; arg: VariablePtr; lastArg: bool const*)

argument	argument type
self	*ast::AstVisitor*
blk	smart_ptr< *ast::ExprBlock* > const
arg	*VariablePtr*
lastArg	bool const

before every block argument

`AstVisitor.`**`visitExprBlockArgument`**(*self: AstVisitor; blk: smart_ptr<ast::ExprBlock> const; arg: VariablePtr; lastArg: bool const*)

visitExprBlockArgument returns *VariablePtr*

argument	argument type
self	*ast::AstVisitor*
blk	smart_ptr< *ast::ExprBlock* > const
arg	*VariablePtr*
lastArg	bool const

after every block argument

`AstVisitor.`**`preVisitExprBlockArgumentInit`** (*self: AstVisitor; blk: smart_ptr<ast::ExprBlock> const; arg: VariablePtr; expr: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
blk	smart_ptr< *ast::ExprBlock* > const
arg	*VariablePtr*
expr	*ExpressionPtr*

before every block argument initialization expression (should it have one), between 'preVisitExprBlockArgument' and *visitExprBlockArgument*

`AstVisitor.`**`visitExprBlockArgumentInit`** (*self: AstVisitor; blk: smart_ptr<ast::ExprBlock> const; arg: VariablePtr; expr: ExpressionPtr*)

visitExprBlockArgumentInit returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
blk	smart_ptr< *ast::ExprBlock* > const
arg	*VariablePtr*
expr	*ExpressionPtr*

after every block argument initialization expression (should it have one), between 'preVisitExprBlockArgument' and *visitExprBlockArgument*

`AstVisitor.`**`preVisitExprBlockExpression`** (*self: AstVisitor; blk: smart_ptr<ast::ExprBlock> const; expr: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
blk	smart_ptr< *ast::ExprBlock* > const
expr	*ExpressionPtr*

before every block expression

`AstVisitor.`**`visitExprBlockExpression`** (*self: AstVisitor; blk: smart_ptr<ast::ExprBlock> const; expr: ExpressionPtr*)

visitExprBlockExpression returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
blk	smart_ptr< *ast::ExprBlock* > const
expr	*ExpressionPtr*

after every block expression

`AstVisitor.`**`preVisitExprBlockFinal`** (*self: AstVisitor; blk: smart_ptr<ast::ExprBlock> const*)

argument	argument type
self	*ast::AstVisitor*
blk	smart_ptr< *ast::ExprBlock* > const

before *finally* ` section of the block

`AstVisitor.`**`visitExprBlockFinal`** (*self: AstVisitor; blk: smart_ptr<ast::ExprBlock> const*)

argument	argument type
self	*ast::AstVisitor*
blk	smart_ptr< *ast::ExprBlock* > const

after *finally* ` section of the block

`AstVisitor.`**`preVisitExprBlockFinalExpression`** (*self: AstVisitor; blk: smart_ptr<ast::ExprBlock> const; expr: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
blk	smart_ptr< *ast::ExprBlock* > const
expr	*ExpressionPtr*

before every block expression in the *finally* section, between *preVisitExprBlockFinal* and *visitExprBlockFinal*

`AstVisitor.`**`visitExprBlockFinalExpression`** (*self: AstVisitor; blk: smart_ptr<ast::ExprBlock> const; expr: ExpressionPtr*)

visitExprBlockFinalExpression returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
blk	smart_ptr< *ast::ExprBlock* > const
expr	*ExpressionPtr*

after every block expression in the *finally* ` section, between *preVisitExprBlockFinal* and *visitExprBlockFinal*

`AstVisitor.`**`preVisitExprLet`**(*self: AstVisitor; expr: smart_ptr<ast::ExprLet> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprLet* > const

before *ExprLet*

`AstVisitor.`**`visitExprLet`**(*self: AstVisitor; expr: smart_ptr<ast::ExprLet> const*)

visitExprLet returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprLet* > const

after *ExprLet*

`AstVisitor.`**`preVisitExprLetVariable`**(*self: AstVisitor; expr: smart_ptr<ast::ExprLet> const; arg: VariablePtr; lastArg: bool const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprLet* > const
arg	*VariablePtr*
lastArg	bool const

before every variable

`AstVisitor.`**`visitExprLetVariable`**(*self: AstVisitor; expr: smart_ptr<ast::ExprLet> const; arg: VariablePtr; lastArg: bool const*) visitExprLetVariable returns *VariablePtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprLet* > const
arg	*VariablePtr*
lastArg	bool const

after every variable

`AstVisitor.`**`preVisitExprLetVariableInit`** (*self: AstVisitor; blk: smart_ptr<ast::ExprLet> const; arg: VariablePtr; expr: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
blk	smart_ptr< *ast::ExprLet* > const
arg	*VariablePtr*
expr	*ExpressionPtr*

before variable initialization (should it have one), between *preVisitExprLetVariable* and *visitExprLetVariable*

`AstVisitor.`**`visitExprLetVariableInit`** (*self: AstVisitor; blk: smart_ptr<ast::ExprLet> const; arg: VariablePtr; expr: ExpressionPtr*)

visitExprLetVariableInit returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
blk	smart_ptr< *ast::ExprLet* > const
arg	*VariablePtr*
expr	*ExpressionPtr*

after variable initialization (should it have one), between *preVisitExprLetVariable* and *visitExprLetVariable*

`AstVisitor.`**`canVisitGlobalVariable`** (*self: AstVisitor; arg: ast::Variable? const*)

canVisitGlobalVariable returns bool

argument	argument type
self	*ast::AstVisitor*
arg	*ast::Variable* ? const

If true global variable declaration will be visited

`AstVisitor.`**`preVisitGlobalLet`** (*self: AstVisitor; prog: ProgramPtr*)

argument	argument type
self	*ast::AstVisitor*
prog	*ProgramPtr*

before global variable declaration

`AstVisitor.`**`visitGlobalLet`** (*self: AstVisitor; prog: ProgramPtr*)

argument	argument type
self	*ast::AstVisitor*
prog	*ProgramPtr*

after global variable declaration

`AstVisitor.`**`preVisitGlobalLetVariable`** (*self: AstVisitor; arg: VariablePtr; lastArg: bool const*)

argument	argument type
self	*ast::AstVisitor*
arg	*VariablePtr*
lastArg	bool const

before every global variable

`AstVisitor.`**`visitGlobalLetVariable`** (*self: AstVisitor; arg: VariablePtr; lastArg: bool const*)

visitGlobalLetVariable returns *VariablePtr*

argument	argument type
self	*ast::AstVisitor*
arg	*VariablePtr*
lastArg	bool const

after every global variable

`AstVisitor.`**`preVisitGlobalLetVariableInit`** (*self: AstVisitor; arg: VariablePtr; expr: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
arg	*VariablePtr*
expr	*ExpressionPtr*

before global variable initialization (should it have one), between *preVisitGlobalLetVariable* and *visitGlobalLetVariable*

`AstVisitor.`**`visitGlobalLetVariableInit`** (*self: AstVisitor; arg: VariablePtr; expr: ExpressionPtr*)

visitGlobalLetVariableInit returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
arg	*VariablePtr*
expr	*ExpressionPtr*

after global variable initialization (should it have one), between *preVisitGlobalLetVariable* and *visitGlobalLetVariable*

`AstVisitor.`**`preVisitExprStringBuilder`** (*self: AstVisitor; expr: smart_ptr<ast::ExprStringBuilder> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprStringBuilder* > const

before *ExprStringBuilder*

AstVisitor.**visitExprStringBuilder** (*self: AstVisitor; expr: smart_ptr<ast::ExprStringBuilder> const*)

visitExprStringBuilder returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprStringBuilder* > const

after *ExprStringBuilder*

AstVisitor.**preVisitExprStringBuilderElement** (*self: AstVisitor; expr: smart_ptr<ast::ExprStringBuilder> const; elem: ExpressionPtr; last: bool const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprStringBuilder* > const
elem	*ExpressionPtr*
last	bool const

before any element of string builder (string or expression)

AstVisitor.**visitExprStringBuilderElement** (*self: AstVisitor; expr: smart_ptr<ast::ExprStringBuilder> const; elem: ExpressionPtr; last: bool const*)

visitExprStringBuilderElement returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprStringBuilder* > const
elem	*ExpressionPtr*
last	bool const

after any element of string builder

AstVisitor.**preVisitExprNew** (*self: AstVisitor; expr: smart_ptr<ast::ExprNew> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprNew* > const

before *ExprNew*

`AstVisitor.`**`visitExprNew`**(*self: AstVisitor; expr: smart_ptr<ast::ExprNew> const*)

visitExprNew returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprNew* > const

after *ExprNew*

`AstVisitor.`**`preVisitExprNewArgument`**(*self: AstVisitor; expr: smart_ptr<ast::ExprNew> const; arg: ExpressionPtr; last: bool const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprNew* > const
arg	*ExpressionPtr*
last	bool const

before every argument

`AstVisitor.`**`visitExprNewArgument`**(*self: AstVisitor; expr: smart_ptr<ast::ExprNew> const; arg: ExpressionPtr; last: bool const*)

visitExprNewArgument returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprNew* > const
arg	*ExpressionPtr*
last	bool const

after every argument

AstVisitor.**preVisitExprNamedCall** (*self: AstVisitor; expr: smart_ptr<ast::ExprNamedCall> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprNamedCall* > const

before *ExprNamedCall*

AstVisitor.**visitExprNamedCall** (*self: AstVisitor; expr: smart_ptr<ast::ExprNamedCall> const*)

visitExprNamedCall returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprNamedCall* > const

after *ExprNamedCall* `

AstVisitor.**preVisitExprNamedCallArgument** (*self: AstVisitor; expr: smart_ptr<ast::ExprNamedCall> const; arg: MakeFieldDeclPtr; last: bool const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprNamedCall* > const
arg	*MakeFieldDeclPtr*
last	bool const

before every argument

AstVisitor.**visitExprNamedCallArgument** (*self: AstVisitor; expr: smart_ptr<ast::ExprNamedCall> const; arg: MakeFieldDeclPtr; last: bool const*)

visitExprNamedCallArgument returns *MakeFieldDeclPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprNamedCall* > const
arg	*MakeFieldDeclPtr*
last	bool const

after every argument

`AstVisitor.`**`preVisitExprLooksLikeCall`**(*self: AstVisitor; expr: smart_ptr<ast::ExprLooksLikeCall> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprLooksLikeCall* > const

before *ExprLooksLikeCall*

`AstVisitor.`**`visitExprLooksLikeCall`**(*self: AstVisitor; expr: smart_ptr<ast::ExprLooksLikeCall> const*)

visitExprLooksLikeCall returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprLooksLikeCall* > const

after *ExprLooksLikeCall*

`AstVisitor.`**`preVisitExprLooksLikeCallArgument`**(*self: AstVisitor; expr: smart_ptr<ast::ExprLooksLikeCall> const; arg: ExpressionPtr; last: bool const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprLooksLikeCall* > const
arg	*ExpressionPtr*
last	bool const

before every argument

`AstVisitor.`**`visitExprLooksLikeCallArgument`**(*self: AstVisitor; expr: smart_ptr<ast::ExprLooksLikeCall> const; arg: ExpressionPtr; last: bool const*)

visitExprLooksLikeCallArgument returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprLooksLikeCall* > const
arg	*ExpressionPtr*
last	bool const

after every argument

`AstVisitor.`**`canVisitCall`**(*self: AstVisitor; expr: ast::ExprCall? const*)

canVisitCall returns bool

argument	argument type
self	*ast::AstVisitor*
expr	*ast::ExprCall* ? const

If false call will be completely skipped, otherwise it behaves normally.

`AstVisitor.`**`preVisitExprCall`**(*self: AstVisitor; expr: smart_ptr<ast::ExprCall> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprCall* > const

before *ExprCall*

`AstVisitor.`**`visitExprCall`**(*self: AstVisitor; expr: smart_ptr<ast::ExprCall> const*)

visitExprCall returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprCall* > const

after *ExprCall*

`AstVisitor.`**`preVisitExprCallArgument`**(*self: AstVisitor; expr: smart_ptr<ast::ExprCall> const; arg: ExpressionPtr; last: bool const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprCall* > const
arg	*ExpressionPtr*
last	bool const

before every argument

`AstVisitor.`**`visitExprCallArgument`**(*self: AstVisitor; expr: smart_ptr<ast::ExprCall> const; arg: ExpressionPtr; last: bool const*)

visitExprCallArgument returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprCall* > const
arg	*ExpressionPtr*
last	bool const

after every argument

`AstVisitor.`**`preVisitExprNullCoalescing`**(*self: AstVisitor; expr: smart_ptr<ast::ExprNullCoalescing> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprNullCoalescing* > const

before *ExprNullCoalescing*

`AstVisitor.`**`visitExprNullCoalescing`**(*self: AstVisitor; expr: smart_ptr<ast::ExprNullCoalescing> const*)

visitExprNullCoalescing returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprNullCoalescing* > const

after *ExprNullCoalescing*

`AstVisitor.`**`preVisitExprNullCoalescingDefault`**(*self: AstVisitor; expr: smart_ptr<ast::ExprNullCoalescing> const; defval: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprNullCoalescing* > const
defval	*ExpressionPtr*

before the default value

`AstVisitor.`**`preVisitExprAt`**(*self: AstVisitor; expr: smart_ptr<ast::ExprAt> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprAt* > const

before *ExprAt*

`AstVisitor.`**`visitExprAt`**(*self: AstVisitor; expr: smart_ptr<ast::ExprAt> const*)

visitExprAt returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprAt* > const

after *ExprAt*

`AstVisitor.`**`preVisitExprAtIndex`**(*self: AstVisitor; expr: smart_ptr<ast::ExprAt> const; index: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprAt* > const
index	*ExpressionPtr*

before the index

`AstVisitor.`**`preVisitExprSafeAt`** (*self: AstVisitor; expr: smart_ptr<ast::ExprSafeAt> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprSafeAt* > const

before *ExprSafeAt*

`AstVisitor.`**`visitExprSafeAt`** (*self: AstVisitor; expr: smart_ptr<ast::ExprSafeAt> const*)

visitExprSafeAt returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprSafeAt* > const

after *ExprSafeAt*

`AstVisitor.`**`preVisitExprSafeAtIndex`** (*self: AstVisitor; expr: smart_ptr<ast::ExprAt> const; index: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprAt* > const
index	*ExpressionPtr*

before the index

`AstVisitor.`**`preVisitExprIs`** (*self: AstVisitor; expr: smart_ptr<ast::ExprIs> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprIs* > const

before *ExprIs*

`AstVisitor.`**`visitExprIs`** (*self: AstVisitor; expr: smart_ptr<ast::ExprIs> const*)

visitExprIs returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprIs* > const

after *ExprIs*

`AstVisitor.`**`preVisitExprIsType`** (*self: AstVisitor; expr: smart_ptr<ast::ExprIs> const; typeDecl: TypeDeclPtr*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprIs* > const
typeDecl	*TypeDeclPtr*

before the type

`AstVisitor.`**`preVisitExprOp2`** (*self: AstVisitor; expr: smart_ptr<ast::ExprOp2> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprOp2* > const

before *ExprOp2*

`AstVisitor.`**`visitExprOp2`** (*self: AstVisitor; expr: smart_ptr<ast::ExprOp2> const*)

visitExprOp2 returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprOp2* > const

after *ExprOp2*

`AstVisitor.`**`preVisitExprOp2Right`** (*self: AstVisitor; expr: smart_ptr<ast::ExprOp2> const; right: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprOp2* > const
right	*ExpressionPtr*

before the right operand

`AstVisitor.`**`preVisitExprOp3`** (*self: AstVisitor; expr: smart_ptr<ast::ExprOp3> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprOp3* > const

before *ExprOp3*

`AstVisitor.`**`visitExprOp3`** (*self: AstVisitor; expr: smart_ptr<ast::ExprOp3> const*)

visitExprOp3 returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprOp3* > const

after *ExprOp3*

`AstVisitor.`**`preVisitExprOp3Left`** (*self: AstVisitor; expr: smart_ptr<ast::ExprOp3> const; left: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprOp3* > const
left	*ExpressionPtr*

before the left option

`AstVisitor.`**`preVisitExprOp3Right`** (*self: AstVisitor; expr: smart_ptr<ast::ExprOp3> const; right: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprOp3* > const
right	*ExpressionPtr*

before the right option

`AstVisitor.`**`preVisitExprCopy`** (*self: AstVisitor; expr: smart_ptr<ast::ExprCopy> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprCopy* > const

before *ExprCopy*

`AstVisitor.`**`visitExprCopy`** (*self: AstVisitor; expr: smart_ptr<ast::ExprCopy> const*)

visitExprCopy returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprCopy* > const

after *ExprCopy*

`AstVisitor.`**`preVisitExprCopyRight`** (*self: AstVisitor; expr: smart_ptr<ast::ExprCopy> const; right: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprCopy* > const
right	*ExpressionPtr*

before the right operand

`AstVisitor.`**`preVisitExprMove`** (*self: AstVisitor; expr: smart_ptr<ast::ExprMove> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMove* > const

before *ExprMove*

`AstVisitor.`**`visitExprMove`** (*self: AstVisitor; expr: smart_ptr<ast::ExprMove> const*)

visitExprMove returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMove* > const

after *ExprMove*

`AstVisitor.`**`preVisitExprMoveRight`** (*self: AstVisitor; expr: smart_ptr<ast::ExprMove> const; right: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMove* > const
right	*ExpressionPtr*

before the right operand

`AstVisitor.`**`preVisitExprClone`** (*self: AstVisitor; expr: smart_ptr<ast::ExprClone> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprClone* > const

before *ExprClone*

`AstVisitor.`**`visitExprClone`**(*self: AstVisitor; expr: smart_ptr<ast::ExprClone> const*)

visitExprClone returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprClone* > const

after *ExprClone*

`AstVisitor.`**`preVisitExprCloneRight`**(*self: AstVisitor; expr: smart_ptr<ast::ExprClone> const; right: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprClone* > const
right	*ExpressionPtr*

before the right operand

`AstVisitor.`**`canVisitWithAliasSubexpression`**(*self: AstVisitor; expr: smart_ptr<ast::ExprAssume> const*)

canVisitWithAliasSubexpression returns bool

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprAssume* > const

before the sub expression in the *ExprAssume*

`AstVisitor.`**`preVisitExprAssume`**(*self: AstVisitor; expr: smart_ptr<ast::ExprAssume> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprAssume* > const

before *ExprAssume*

`AstVisitor.`**`visitExprAssume`**(*self: AstVisitor; expr: smart_ptr<ast::ExprAssume> const*)

visitExprAssume returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprAssume* > const

after *ExprAssume*

`AstVisitor.`**`preVisitExprWith`**(*self: AstVisitor; expr: smart_ptr<ast::ExprWith> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprWith* > const

before *ExprWith*

`AstVisitor.`**`visitExprWith`**(*self: AstVisitor; expr: smart_ptr<ast::ExprWith> const*)

visitExprWith returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprWith* > const

after *ExprWith*

`AstVisitor.`**`preVisitExprWithBody`**(*self: AstVisitor; expr: smart_ptr<ast::ExprWith> const; right: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprWith* > const
right	*ExpressionPtr*

before the body block

`AstVisitor.`**`preVisitExprWhile`**(*self: AstVisitor; expr: smart_ptr<ast::ExprWhile> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprWhile* > const

before *ExprWhile*

`AstVisitor.`**`visitExprWhile`**(*self: AstVisitor; expr: smart_ptr<ast::ExprWhile> const*)

visitExprWhile returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprWhile* > const

after *ExprWhile*

`AstVisitor.`**`preVisitExprWhileBody`**(*self: AstVisitor; expr: smart_ptr<ast::ExprWhile> const; right: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprWhile* > const
right	*ExpressionPtr*

before the body block

`AstVisitor.`**`preVisitExprTryCatch`**(*self: AstVisitor; expr: smart_ptr<ast::ExprTryCatch> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprTryCatch* > const

before *ExprTryCatch*

`AstVisitor.`**`visitExprTryCatch`**(*self: AstVisitor; expr: smart_ptr<ast::ExprTryCatch> const*)

visitExprTryCatch returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprTryCatch* > const

after *ExprTryCatch*

`AstVisitor.`**`preVisitExprTryCatchCatch`**(*self: AstVisitor; expr: smart_ptr<ast::ExprTryCatch> const; right: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprTryCatch* > const
right	*ExpressionPtr*

before the catch (recover) section

`AstVisitor.`**`preVisitExprIfThenElse`**(*self: AstVisitor; expr: smart_ptr<ast::ExprIfThenElse> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprIfThenElse* > const

before *ExprIfThenElse*

`AstVisitor.`**`visitExprIfThenElse`**(*self: AstVisitor; expr: smart_ptr<ast::ExprIfThenElse> const*)

visitExprIfThenElse returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprIfThenElse* > const

after *ExprIfThenElse*

`AstVisitor.`**`preVisitExprIfThenElseIfBlock`**(*self: AstVisitor; expr: smart_ptr<ast::ExprIfThenElse> const; ifBlock: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprIfThenElse* > const
ifBlock	*ExpressionPtr*

before the if block

`AstVisitor.`**`preVisitExprIfThenElseElseBlock`**(*self: AstVisitor; expr: smart_ptr<ast::ExprIfThenElse> const; elseBlock: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprIfThenElse* > const
elseBlock	*ExpressionPtr*

before the else block

`AstVisitor.`**`preVisitExprFor`**(*self: AstVisitor; expr: smart_ptr<ast::ExprFor> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprFor* > const

before the *ExprFor*

`AstVisitor.`**`visitExprFor`**(*self: AstVisitor; expr: smart_ptr<ast::ExprFor> const*)

visitExprFor returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprFor* > const

after the *ExprFor*

`AstVisitor.`**`preVisitExprForVariable`**(*self: AstVisitor; expr: smart_ptr<ast::ExprFor> const; svar: VariablePtr; last: bool const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprFor* > const
svar	*VariablePtr*
last	bool const

before each variable

`AstVisitor.`**`visitExprForVariable`**(*self: AstVisitor; expr: smart_ptr<ast::ExprFor> const; svar: VariablePtr; last: bool const*)

visitExprForVariable returns *VariablePtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprFor* > const
svar	*VariablePtr*
last	bool const

after each variable

`AstVisitor.`**`preVisitExprForSource`**(*self: AstVisitor; expr: smart_ptr<ast::ExprFor> const; source: ExpressionPtr; last: bool const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprFor* > const
source	*ExpressionPtr*
last	bool const

before each source

`AstVisitor.`**`visitExprForSource`**(*self: AstVisitor; expr: smart_ptr<ast::ExprFor> const; source: ExpressionPtr; last: bool const*)

visitExprForSource returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprFor* > const
source	*ExpressionPtr*
last	bool const

after each source

`AstVisitor.`**`preVisitExprForStack`**(*self: AstVisitor; expr: smart_ptr<ast::ExprFor> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprFor* > const

before the stack is allocated before the body, regardless if it has one

`AstVisitor.`**`preVisitExprForBody`**(*self: AstVisitor; expr: smart_ptr<ast::ExprFor> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprFor* > const

before the body (should it have one)

AstVisitor.**preVisitExprMakeVariant** (*self: AstVisitor; expr: smart_ptr<ast::ExprMakeVariant> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeVariant* > const

before *ExprMakeVariant*

AstVisitor.**visitExprMakeVariant** (*self: AstVisitor; expr: smart_ptr<ast::ExprMakeVariant> const*)

visitExprMakeVariant returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeVariant* > const

after *ExprMakeVariant*

AstVisitor.**preVisitExprMakeVariantField** (*self: AstVisitor; expr: smart_ptr<ast::ExprMakeVariant> const; index: int const; decl: MakeFieldDeclPtr; last: bool const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeVariant* > const
index	int const
decl	*MakeFieldDeclPtr*
last	bool const

before every field

AstVisitor.**visitExprMakeVariantField** (*self: AstVisitor; expr: smart_ptr<ast::ExprMakeVariant> const; index: int const; decl: MakeFieldDeclPtr; last: bool const*)

visitExprMakeVariantField returns *MakeFieldDeclPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeVariant* > const
index	int const
decl	*MakeFieldDeclPtr*
last	bool const

after every field

`AstVisitor.`**`canVisitMakeStructBody`**(*self: AstVisitor; expr: smart_ptr<ast::ExprMakeStruct> const*)

canVisitMakeStructBody returns bool

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeStruct* > const

if true the visitor can visit the body of *ExprMakeStruct*

`AstVisitor.`**`canVisitMakeStructBlock`**(*self: AstVisitor; expr: smart_ptr<ast::ExprMakeStruct> const; blk: ExpressionPtr*)

canVisitMakeStructBlock returns bool

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeStruct* > const
blk	*ExpressionPtr*

if true the visitor can visit the block behind *ExprMakeStruct*

`AstVisitor.`**`preVisitExprMakeStruct`**(*self: AstVisitor; expr: smart_ptr<ast::ExprMakeStruct> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeStruct* > const

before *ExprMakeStruct*

`AstVisitor.visitExprMakeStruct` (*self: AstVisitor; expr: smart_ptr<ast::ExprMakeStruct> const*)

visitExprMakeStruct returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeStruct* > const

after *ExprMakeStruct*

`AstVisitor.preVisitExprMakeStructIndex` (*self: AstVisitor; expr: smart_ptr<ast::ExprMakeStruct> const; index: int const; last: bool const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeStruct* > const
index	int const
last	bool const

before each struct in the array of structures

`AstVisitor.visitExprMakeStructIndex` (*self: AstVisitor; expr: smart_ptr<ast::ExprMakeStruct> const; index: int const; last: bool const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeStruct* > const
index	int const
last	bool const

after each struct in the array of structures

`AstVisitor.preVisitExprMakeStructField` (*self: AstVisitor; expr: smart_ptr<ast::ExprMakeStruct> const; index: int const; decl: MakeFieldDeclPtr; last: bool const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeStruct* > const
index	int const
decl	*MakeFieldDeclPtr*
last	bool const

before each field of the struct, between *preVisitExprMakeStructIndex* and *visitExprMakeStructIndex*

AstVisitor.**visitExprMakeStructField**(*self: AstVisitor; expr: smart_ptr<ast::ExprMakeStruct> const; index: int const; decl: MakeFieldDeclPtr; last: bool const*)

visitExprMakeStructField returns *MakeFieldDeclPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeStruct* > const
index	int const
decl	*MakeFieldDeclPtr*
last	bool const

after each field of the struct, between *preVisitExprMakeStructIndex* and *visitExprMakeStructIndex*

AstVisitor.**preVisitExprMakeArray**(*self: AstVisitor; expr: smart_ptr<ast::ExprMakeArray> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeArray* > const

before *ExprMakeArray*

AstVisitor.**visitExprMakeArray**(*self: AstVisitor; expr: smart_ptr<ast::ExprMakeArray> const*)

visitExprMakeArray returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeArray* > const

after *ExprMakeArray*

`AstVisitor.`**`preVisitExprMakeArrayIndex`**(*self: AstVisitor; expr: smart_ptr<ast::ExprMakeArray> const; index: int const; init: ExpressionPtr; last: bool const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeArray* > const
index	int const
init	*ExpressionPtr*
last	bool const

before each element of the array

`AstVisitor.`**`visitExprMakeArrayIndex`**(*self: AstVisitor; expr: smart_ptr<ast::ExprMakeArray> const; index: int const; init: ExpressionPtr; last: bool const*)

visitExprMakeArrayIndex returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeArray* > const
index	int const
init	*ExpressionPtr*
last	bool const

after each element of the array

`AstVisitor.`**`preVisitExprMakeTuple`**(*self: AstVisitor; expr: smart_ptr<ast::ExprMakeTuple> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeTuple* > const

before *ExprMakeTuple*

`AstVisitor.`**`visitExprMakeTuple`**(*self: AstVisitor; expr: smart_ptr<ast::ExprMakeTuple> const*)

visitExprMakeTuple returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeTuple* > const

after *ExprMakeTuple*

`AstVisitor.`**`preVisitExprMakeTupleIndex`**(*self: AstVisitor; expr: smart_ptr<ast::ExprMakeTuple> const; index: int const; init: ExpressionPtr; last: bool const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeTuple* > const
index	int const
init	*ExpressionPtr*
last	bool const

before each field of the tuple

`AstVisitor.`**`visitExprMakeTupleIndex`**(*self: AstVisitor; expr: smart_ptr<ast::ExprMakeTuple> const; index: int const; init: ExpressionPtr; last: bool const*)

visitExprMakeTupleIndex returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeTuple* > const
index	int const
init	*ExpressionPtr*
last	bool const

after each field of the tuple

`AstVisitor.`**`preVisitExprArrayComprehension`**(*self: AstVisitor; expr: smart_ptr<ast::ExprArrayComprehension> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprArrayComprehension* > const

before *ExprArrayComprehension*

`AstVisitor.`**`visitExprArrayComprehension`**(*self: AstVisitor; expr: smart_ptr<ast::ExprArrayComprehension> const*)

visitExprArrayComprehension returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprArrayComprehension* > const

after *ExprArrayComprehension*

`AstVisitor.`**`preVisitExprArrayComprehensionSubexpr`**(*self: AstVisitor; expr: smart_ptr<ast::ExprArrayComprehension> const; subexrp: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprArrayComprehension* > const
subexrp	*ExpressionPtr*

before the subexpression

`AstVisitor.`**`preVisitExprArrayComprehensionWhere`**(*self: AstVisitor; expr: smart_ptr<ast::ExprArrayComprehension> const; filter: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprArrayComprehension* > const
filter	*ExpressionPtr*

before the where clause

`AstVisitor.`**`preVisitExprTypeInfo`**(*self: AstVisitor; expr: smart_ptr<ast::ExprTypeInfo> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprTypeInfo* > const

before *ExprTypeInfo*

`AstVisitor.`**`visitExprTypeInfo`**(*self: AstVisitor; expr: smart_ptr<ast::ExprTypeInfo> const*)

visitExprTypeInfo returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprTypeInfo* > const

after *ExprTypeInfo*

`AstVisitor.`**`preVisitExprPtr2Ref`**(*self: AstVisitor; expr: smart_ptr<ast::ExprPtr2Ref> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprPtr2Ref* > const

before *ExprPtr2Ref*

`AstVisitor.`**`visitExprPtr2Ref`**(*self: AstVisitor; expr: smart_ptr<ast::ExprPtr2Ref> const*)

visitExprPtr2Ref returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprPtr2Ref* > const

after *ExprPtr2Ref*

`AstVisitor.`**`preVisitExprLabel`**(*self: AstVisitor; expr: smart_ptr<ast::ExprLabel> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprLabel* > const

before *ExprLabel*

`AstVisitor.`**`visitExprLabel`**(*self: AstVisitor; expr: smart_ptr<ast::ExprLabel> const*)

visitExprLabel returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprLabel* > const

after *ExprLabel*

`AstVisitor.`**`preVisitExprGoto`**(*self: AstVisitor; expr: smart_ptr<ast::ExprGoto> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprGoto* > const

before *ExprGoto*

`AstVisitor.`**`visitExprGoto`**(*self: AstVisitor; expr: smart_ptr<ast::ExprGoto> const*)

visitExprGoto returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprGoto* > const

after *ExprGoto*

`AstVisitor.`**`preVisitExprRef2Value`**(*self: AstVisitor; expr: smart_ptr<ast::ExprRef2Value> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprRef2Value* > const

before *ExprRef2Value*

`AstVisitor.`**`visitExprRef2Value`**(*self: AstVisitor; expr: smart_ptr<ast::ExprRef2Value> const*)

visitExprRef2Value returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprRef2Value* > const

after *ExprRef2Value*

`AstVisitor.`**`preVisitExprRef2Ptr`**(*self: AstVisitor; expr: smart_ptr<ast::ExprRef2Ptr> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprRef2Ptr* > const

before *ExprRef2Ptr*

`AstVisitor.`**`visitExprRef2Ptr`**(*self: AstVisitor; expr: smart_ptr<ast::ExprRef2Ptr> const*)

visitExprRef2Ptr returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprRef2Ptr* > const

after *ExprRef2Ptr*

`AstVisitor.`**`preVisitExprAddr`**(*self: AstVisitor; expr: smart_ptr<ast::ExprAddr> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprAddr* > const

before *ExprAddr*

`AstVisitor.`**`visitExprAddr`**(*self: AstVisitor; expr: smart_ptr<ast::ExprAddr> const*)

visitExprAddr returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprAddr* > const

after *ExprAddr*

`AstVisitor.`**`preVisitExprAssert`**(*self: AstVisitor; expr: smart_ptr<ast::ExprAssert> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprAssert* > const

before *ExprAssert*

`AstVisitor.`**`visitExprAssert`**(*self: AstVisitor; expr: smart_ptr<ast::ExprAssert> const*)

visitExprAssert returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprAssert* > const

after *ExprAssert*

`AstVisitor.`**`preVisitExprStaticAssert`**(*self: AstVisitor; expr: smart_ptr<ast::ExprStaticAssert> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprStaticAssert* > const

before *ExprStaticAssert*

`AstVisitor.`**`visitExprStaticAssert`**(*self: AstVisitor; expr: smart_ptr<ast::ExprStaticAssert> const*)

visitExprStaticAssert returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprStaticAssert* > const

after *ExprStaticAssert*

`AstVisitor.`**`preVisitExprQuote`**(*self: AstVisitor; expr: smart_ptr<ast::ExprQuote> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprQuote* > const

before *ExprQuote*

`AstVisitor.`**`visitExprQuote`**(*self: AstVisitor; expr: smart_ptr<ast::ExprQuote> const*)

visitExprQuote returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprQuote* > const

after *ExprQuote*

`AstVisitor.`**`preVisitExprDebug`**(*self: AstVisitor; expr: smart_ptr<ast::ExprDebug> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprDebug* > const

before *ExprDebug*

`AstVisitor.`**`visitExprDebug`**(*self: AstVisitor; expr: smart_ptr<ast::ExprDebug> const*)

visitExprDebug returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprDebug* > const

after *ExprDebug*

`AstVisitor.`**`preVisitExprInvoke`**(*self: AstVisitor; expr: smart_ptr<ast::ExprInvoke> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprInvoke* > const

before *ExprInvoke*

`AstVisitor.`**`visitExprInvoke`**(*self: AstVisitor; expr: smart_ptr<ast::ExprInvoke> const*)

visitExprInvoke returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprInvoke* > const

after *ExprInvoke*

`AstVisitor.`**`preVisitExprErase`**(*self: AstVisitor; expr: smart_ptr<ast::ExprErase> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprErase* > const

before *ExprErase*

`AstVisitor.`**`visitExprErase`**(*self: AstVisitor; expr: smart_ptr<ast::ExprErase> const*)

visitExprErase returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprErase* > const

after *ExprErase*

AstVisitor.**preVisitExprSetInsert** (*self: AstVisitor; expr: smart_ptr<ast::ExprSetInsert> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprSetInsert* > const

before *ExprSetInsert*

AstVisitor.**visitExprSetInsert** (*self: AstVisitor; expr: smart_ptr<ast::ExprSetInsert> const*)

visitExprSetInsert returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprSetInsert* > const

after *ExprSetInsert*

AstVisitor.**preVisitExprFind** (*self: AstVisitor; expr: smart_ptr<ast::ExprFind> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprFind* > const

before *ExprFind*

AstVisitor.**visitExprFind** (*self: AstVisitor; expr: smart_ptr<ast::ExprFind> const*)

visitExprFind returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprFind* > const

after *ExprFind*

AstVisitor.**preVisitExprKeyExists** (*self: AstVisitor; expr: smart_ptr<ast::ExprKeyExists> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprKeyExists* > const

before *ExprKeyExists*

`AstVisitor.`**`visitExprKeyExists`**(*self: AstVisitor; expr: smart_ptr<ast::ExprKeyExists> const*)

visitExprKeyExists returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprKeyExists* > const

after *ExprKeyExists*

`AstVisitor.`**`preVisitExprAscend`**(*self: AstVisitor; expr: smart_ptr<ast::ExprAscend> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprAscend* > const

before *ExprAscend*

`AstVisitor.`**`visitExprAscend`**(*self: AstVisitor; expr: smart_ptr<ast::ExprAscend> const*)

visitExprAscend returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprAscend* > const

after *ExprAscend*

`AstVisitor.`**`preVisitExprCast`**(*self: AstVisitor; expr: smart_ptr<ast::ExprCast> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprCast* > const

before *ExprCast*

`AstVisitor.`**`visitExprCast`** (*self: AstVisitor; expr: smart_ptr<ast::ExprCast> const*)

visitExprCast returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprCast* > const

after *ExprCast*

`AstVisitor.`**`preVisitExprDelete`** (*self: AstVisitor; expr: smart_ptr<ast::ExprDelete> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprDelete* > const

before *ExprDelete*

`AstVisitor.`**`visitExprDelete`** (*self: AstVisitor; expr: smart_ptr<ast::ExprDelete> const*)

visitExprDelete returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprDelete* > const

after *ExprDelete*

`AstVisitor.`**`preVisitExprVar`** (*self: AstVisitor; expr: smart_ptr<ast::ExprVar> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprVar* > const

before *ExprVar*

`AstVisitor.`**`visitExprVar`** (*self: AstVisitor; expr: smart_ptr<ast::ExprVar> const*)

visitExprVar returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprVar* > const

after *ExprVar*

`AstVisitor.`**`preVisitExprTag`**(*self: AstVisitor; expr: smart_ptr<ast::ExprTag> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprTag* > const

before *ExprTag*

`AstVisitor.`**`preVisitExprTagValue`**(*self: AstVisitor; expr: smart_ptr<ast::ExprTag> const; value: ExpressionPtr*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprTag* > const
value	*ExpressionPtr*

before the value portion of *ExprTag*

`AstVisitor.`**`visitExprTag`**(*self: AstVisitor; expr: smart_ptr<ast::ExprTag> const*)

visitExprTag returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprTag* > const

after *ExprTag*

`AstVisitor.`**`preVisitExprField`**(*self: AstVisitor; expr: smart_ptr<ast::ExprField> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprField* > const

before *ExprField*

AstVisitor.**visitExprField**(*self: AstVisitor; expr: smart_ptr<ast::ExprField> const*)

visitExprField returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprField* > const

after *ExprField*

AstVisitor.**preVisitExprSafeField**(*self: AstVisitor; expr: smart_ptr<ast::ExprSafeField> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprSafeField* > const

before *ExprSafeField*

AstVisitor.**visitExprSafeField**(*self: AstVisitor; expr: smart_ptr<ast::ExprSafeField> const*)

visitExprSafeField returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprSafeField* > const

after *ExprSafeField*

AstVisitor.**preVisitExprSwizzle**(*self: AstVisitor; expr: smart_ptr<ast::ExprSwizzle> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprSwizzle* > const

before *ExprSwizzle*

AstVisitor.**visitExprSwizzle**(*self: AstVisitor; expr: smart_ptr<ast::ExprSwizzle> const*)

visitExprSwizzle returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprSwizzle* > const

after *ExprSwizzle*

`AstVisitor.`**`preVisitExprIsVariant`** (*self: AstVisitor; expr: smart_ptr<ast::ExprIsVariant> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprIsVariant* > const

before *ExprIsVariant*

`AstVisitor.`**`visitExprIsVariant`** (*self: AstVisitor; expr: smart_ptr<ast::ExprIsVariant> const*)

visitExprIsVariant returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprIsVariant* > const

after *ExprIsVariant*

`AstVisitor.`**`preVisitExprAsVariant`** (*self: AstVisitor; expr: smart_ptr<ast::ExprAsVariant> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprAsVariant* > const

before *ExprAsVariant*

`AstVisitor.`**`visitExprAsVariant`** (*self: AstVisitor; expr: smart_ptr<ast::ExprAsVariant> const*)

visitExprAsVariant returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprAsVariant* > const

after *ExprAsVariant*

AstVisitor.**preVisitExprSafeAsVariant** (*self: AstVisitor; expr: smart_ptr<ast::ExprSafeAsVariant> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprSafeAsVariant* > const

before *ExprSafeAsVariant*

AstVisitor.**visitExprSafeAsVariant** (*self: AstVisitor; expr: smart_ptr<ast::ExprSafeAsVariant> const*)

visitExprSafeAsVariant returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprSafeAsVariant* > const

after *ExprSafeAsVariant*

AstVisitor.**preVisitExprOp1** (*self: AstVisitor; expr: smart_ptr<ast::ExprOp1> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprOp1* > const

before *ExprOp1*

AstVisitor.**visitExprOp1** (*self: AstVisitor; expr: smart_ptr<ast::ExprOp1> const*)

visitExprOp1 returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprOp1* > const

after *ExprOp1*

AstVisitor.**preVisitExprReturn** (*self: AstVisitor; expr: smart_ptr<ast::ExprReturn> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprReturn* > const

before *ExprReturn*

AstVisitor.**visitExprReturn**(*self: AstVisitor; expr: smart_ptr<ast::ExprReturn> const*)

visitExprReturn returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprReturn* > const

after *ExprReturn*

AstVisitor.**preVisitExprYield**(*self: AstVisitor; expr: smart_ptr<ast::ExprYield> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprYield* > const

before *ExprYield*

AstVisitor.**visitExprYield**(*self: AstVisitor; expr: smart_ptr<ast::ExprYield> const*)

visitExprYield returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprYield* > const

after 'ExprYield'

AstVisitor.**preVisitExprBreak**(*self: AstVisitor; expr: smart_ptr<ast::ExprBreak> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprBreak* > const

before *ExprBreak*

`AstVisitor.`**`visitExprBreak`** (*self: AstVisitor; expr: smart_ptr<ast::ExprBreak> const*)

visitExprBreak returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprBreak* > const

after *ExprBreak*

`AstVisitor.`**`preVisitExprContinue`** (*self: AstVisitor; expr: smart_ptr<ast::ExprContinue> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprContinue* > const

before *ExprContinue*

`AstVisitor.`**`visitExprContinue`** (*self: AstVisitor; expr: smart_ptr<ast::ExprContinue> const*)

visitExprContinue returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprContinue* > const

after *ExprContinue*

`AstVisitor.`**`canVisitMakeBlockBody`** (*self: AstVisitor; expr: smart_ptr<ast::ExprMakeBlock> const*)

canVisitMakeBlockBody returns bool

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeBlock* > const

before the body of the *makeBlock* expression is visited. If true *body* will be visited

`AstVisitor.`**`preVisitExprMakeBlock`** (*self: AstVisitor; expr: smart_ptr<ast::ExprMakeBlock> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeBlock* > const

before *ExprMakeBlock*

`AstVisitor.`**`visitExprMakeBlock`**(*self: AstVisitor; expr: smart_ptr<ast::ExprMakeBlock> const*)

visitExprMakeBlock returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeBlock* > const

after *ExprMakeBlock*

`AstVisitor.`**`preVisitExprMakeGenerator`**(*self: AstVisitor; expr: smart_ptr<ast::ExprMakeGenerator> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeGenerator* > const

before *ExprMakeGenerator*

`AstVisitor.`**`visitExprMakeGenerator`**(*self: AstVisitor; expr: smart_ptr<ast::ExprMakeGenerator> const*)

visitExprMakeGenerator returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMakeGenerator* > const

after *ExprMakeGenerator*

`AstVisitor.`**`preVisitExprMemZero`**(*self: AstVisitor; expr: smart_ptr<ast::ExprMemZero> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMemZero* > const

before *ExprMemZero*

`AstVisitor.`**`visitExprMemZero`**(*self: AstVisitor; expr: smart_ptr<ast::ExprMemZero> const*)

visitExprMemZero returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprMemZero* > const

after *ExprMemZero*

`AstVisitor.`**`preVisitExprConst`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConst> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConst* > const

before *ExprConst*

`AstVisitor.`**`visitExprConst`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConst> const*)

visitExprConst returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConst* > const

after *ExprConst*

`AstVisitor.`**`preVisitExprConstPtr`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstPtr> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstPtr* > const

before *ExprConstPtr*

`AstVisitor.`**`visitExprConstPtr`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstPtr> const*)

visitExprConstPtr returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstPtr* > const

after *ExprConstPtr*

`AstVisitor.`**`preVisitExprConstEnumeration`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstEnumeration> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstEnumeration* > const

before *ExprConstEnumeration*

`AstVisitor.`**`visitExprConstEnumeration`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstEnumeration> const*)

visitExprConstEnumeration returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstEnumeration* > const

after *ExprConstEnumeration*

`AstVisitor.`**`preVisitExprConstBitfield`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstBitfield> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstBitfield* > const

before *ExprConstBitfield*

`AstVisitor.`**`visitExprConstBitfield`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstBitfield> const*) visitExprConstBitfield returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstBitfield* > const

after *ExprConstBitfield*

`AstVisitor.`**`preVisitExprConstInt8`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstInt8> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstInt8* > const

before *ExprConstInt8*

`AstVisitor.`**`visitExprConstInt8`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstInt8> const*)

visitExprConstInt8 returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstInt8* > const

after *ExprConstInt8*

`AstVisitor.`**`preVisitExprConstInt16`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstInt16> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstInt16* > const

before *ExprConstInt16*

`AstVisitor.`**`visitExprConstInt16`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstInt16> const*)

visitExprConstInt16 returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstInt16* > const

after *ExprConstInt16*

`AstVisitor.`**`preVisitExprConstInt64`** (*self: AstVisitor; expr: smart_ptr<ast::ExprConstInt64> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstInt64* > const

before *ExprConstInt64*

`AstVisitor.`**`visitExprConstInt64`** (*self: AstVisitor; expr: smart_ptr<ast::ExprConstInt64> const*)

visitExprConstInt64 returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstInt64* > const

after *ExprConstInt64*

`AstVisitor.`**`preVisitExprConstInt`** (*self: AstVisitor; expr: smart_ptr<ast::ExprConstInt> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstInt* > const

before *ExprConstInt*

`AstVisitor.`**`visitExprConstInt`** (*self: AstVisitor; expr: smart_ptr<ast::ExprConstInt> const*)

visitExprConstInt returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstInt* > const

after *ExprConstInt*

`AstVisitor.`**`preVisitExprConstInt2`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstInt2> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstInt2* > const

before *ExprConstInt2*

`AstVisitor.`**`visitExprConstInt2`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstInt2> const*)

visitExprConstInt2 returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstInt2* > const

after *ExprConstInt2*

`AstVisitor.`**`preVisitExprConstInt3`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstInt3> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstInt3* > const

before *ExprConstInt3*

`AstVisitor.`**`visitExprConstInt3`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstInt3> const*)

visitExprConstInt3 returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstInt3* > const

after *ExprConstInt3*

`AstVisitor.`**`preVisitExprConstInt4`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstInt4> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstInt4* > const

before *ExprConstInt4*

`AstVisitor.`**`visitExprConstInt4`** (*self: AstVisitor; expr: smart_ptr<ast::ExprConstInt4> const*)

visitExprConstInt4 returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstInt4* > const

after *ExprConstInt4*

`AstVisitor.`**`preVisitExprConstUInt8`** (*self: AstVisitor; expr: smart_ptr<ast::ExprConstUInt8> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstUInt8* > const

before *ExprConstUInt8*

`AstVisitor.`**`visitExprConstUInt8`** (*self: AstVisitor; expr: smart_ptr<ast::ExprConstUInt8> const*)

visitExprConstUInt8 returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstUInt8* > const

after *ExprConstUInt8*

`AstVisitor.`**`preVisitExprConstUInt16`** (*self: AstVisitor; expr: smart_ptr<ast::ExprConstUInt16> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstUInt16* > const

before *ExprConstUInt16*

`AstVisitor.`**`visitExprConstUInt16`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstUInt16> const*)

visitExprConstUInt16 returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstUInt16* > const

after *ExprConstUInt16*

`AstVisitor.`**`preVisitExprConstUInt64`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstUInt64> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstUInt64* > const

before *ExprConstUInt64*

`AstVisitor.`**`visitExprConstUInt64`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstUInt64> const*)

visitExprConstUInt64 returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstUInt64* > const

after *ExprConstUInt64*

`AstVisitor.`**`preVisitExprConstUInt`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstUInt> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstUInt* > const

before *ExprConstUInt*

`AstVisitor.`**`visitExprConstUInt`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstUInt> const*) visitExprConstUInt returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstUInt* > const

after *ExprConstUInt*

`AstVisitor.`**`preVisitExprConstUInt2`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstUInt2> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstUInt2* > const

before *ExprConstUInt2*

`AstVisitor.`**`visitExprConstUInt2`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstUInt2> const*)

visitExprConstUInt2 returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstUInt2* > const

after *ExprConstUInt2*

`AstVisitor.`**`preVisitExprConstUInt3`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstUInt3> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstUInt3* > const

before *ExprConstUInt3*

`AstVisitor.`**`visitExprConstUInt3`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstUInt3> const*)

visitExprConstUInt3 returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstUInt3* > const

after *ExprConstUInt3*

`AstVisitor.`**`preVisitExprConstUInt4`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstUInt4> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstUInt4* > const

before *ExprConstUInt4*

`AstVisitor.`**`visitExprConstUInt4`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstUInt4> const*)

visitExprConstUInt4 returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstUInt4* > const

after *ExprConstUInt4*

`AstVisitor.`**`preVisitExprConstRange`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstRange> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstRange* > const

before *ExprConstRange*

`AstVisitor.`**`visitExprConstRange`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstRange> const*)

visitExprConstRange returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstRange* > const

after *ExprConstRange*

`AstVisitor.`**`preVisitExprConstURange`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstURange> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstURange* > const

before *ExprConstURange*

`AstVisitor.`**`visitExprConstURange`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstURange> const*)

visitExprConstURange returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstURange* > const

after *ExprConstURange*

`AstVisitor.`**`preVisitExprConstRange64`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstRange64> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstRange64* > const

before *ExprConstRange64*

`AstVisitor.`**`visitExprConstRange64`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstRange64> const*)

visitExprConstRange64 returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstRange64* > const

after *ExprConstRange64*

`AstVisitor.`**`preVisitExprConstURange64`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstURange64> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstURange64* > const

before *ExprConstURange64*

`AstVisitor.`**`visitExprConstURange64`** (*self: AstVisitor; expr: smart_ptr<ast::ExprConstURange64> const*)

visitExprConstURange64 returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstURange64* > const

after *ExprConstURange64*

`AstVisitor.`**`preVisitExprConstBool`** (*self: AstVisitor; expr: smart_ptr<ast::ExprConstBool> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstBool* > const

before *ExprConstBool*

`AstVisitor.`**`visitExprConstBool`** (*self: AstVisitor; expr: smart_ptr<ast::ExprConstBool> const*)

visitExprConstBool returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstBool* > const

after *ExprConstBool*

`AstVisitor.`**`preVisitExprConstFloat`** (*self: AstVisitor; expr: smart_ptr<ast::ExprConstFloat> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstFloat* > const

before *ExprConstFloat*

`AstVisitor.`**`visitExprConstFloat`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstFloat> const*)

visitExprConstFloat returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstFloat* > const

after *ExprConstFloat*

`AstVisitor.`**`preVisitExprConstFloat2`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstFloat2> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstFloat2* > const

before *ExprConstFloat2*

`AstVisitor.`**`visitExprConstFloat2`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstFloat2> const*)

visitExprConstFloat2 returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstFloat2* > const

after *ExprConstFloat2*

`AstVisitor.`**`preVisitExprConstFloat3`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstFloat3> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstFloat3* > const

before *ExprConstFloat3*

`AstVisitor.`**`visitExprConstFloat3`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstFloat3> const*)

visitExprConstFloat3 returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstFloat3* > const

after *ExprConstFloat3*

`AstVisitor.`**`preVisitExprConstFloat4`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstFloat4> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstFloat4* > const

before *ExprConstFloat4*

`AstVisitor.`**`visitExprConstFloat4`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstFloat4> const*)

visitExprConstFloat4 returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstFloat4* > const

after *ExprConstFloat4*

`AstVisitor.`**`preVisitExprConstString`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstString> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstString* > const

before *ExprConstString*

`AstVisitor.`**`visitExprConstString`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstString> const*)

visitExprConstString returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstString* > const

after *ExprConstString*

`AstVisitor.`**`preVisitExprConstDouble`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstDouble> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstDouble* > const

before *ExprConstDouble*

`AstVisitor.`**`visitExprConstDouble`**(*self: AstVisitor; expr: smart_ptr<ast::ExprConstDouble> const*)

visitExprConstDouble returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprConstDouble* > const

after *ExprConstDouble*

`AstVisitor.`**`preVisitExprFakeContext`**(*self: AstVisitor; expr: smart_ptr<ast::ExprFakeContext> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprFakeContext* > const

before *ExprConstFakeContext*

`AstVisitor.`**`visitExprFakeContext`**(*self: AstVisitor; expr: smart_ptr<ast::ExprFakeContext> const*)

visitExprFakeContext returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprFakeContext* > const

after *ExprConstFakeContext*

`AstVisitor.`**`preVisitExprFakeLineInfo`**(*self: AstVisitor; expr: smart_ptr<ast::ExprFakeLineInfo> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprFakeLineInfo* > const

before *ExprConstFakeLineInfo*

`AstVisitor.`**`visitExprFakeLineInfo`**(*self: AstVisitor; expr: smart_ptr<ast::ExprFakeLineInfo> const*)

visitExprFakeLineInfo returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprFakeLineInfo* > const

after *ExprConstFakeLineInfo*

`AstVisitor.`**`preVisitExprReader`**(*self: AstVisitor; expr: smart_ptr<ast::ExprReader> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprReader* > const

before *ExprReader*

`AstVisitor.`**`visitExprReader`**(*self: AstVisitor; expr: smart_ptr<ast::ExprReader> const*)

visitExprReader returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprReader* > const

after *ExprReader*

`AstVisitor.`**`preVisitExprUnsafe`**(*self: AstVisitor; expr: smart_ptr<ast::ExprUnsafe> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprUnsafe* > const

before *ExprUnsafe*

`AstVisitor.`**`visitExprUnsafe`**(*self: AstVisitor; expr: smart_ptr<ast::ExprUnsafe> const*)

visitExprUnsafe returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprUnsafe* > const

after *ExprUnsafe*

`AstVisitor.`**`preVisitExprCallMacro`**(*self: AstVisitor; expr: smart_ptr<ast::ExprCallMacro> const*)

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprCallMacro* > const

before *ExprCallMacro*

`AstVisitor.`**visitExprCallMacro**(*self: AstVisitor; expr: smart_ptr<ast::ExprCallMacro> const*)

visitExprCallMacro returns *ExpressionPtr*

argument	argument type
self	*ast::AstVisitor*
expr	smart_ptr< *ast::ExprCallMacro* > const

after *ExprCallMacro*

12.8 Call generation

- *make_call (at:rtti::LineInfo const implicit;name:string const implicit) : smart_ptr<ast::Expression>*

make_call(*at: LineInfo const implicit; name: string const implicit*)

make_call returns smart_ptr< *ast::Expression* >

argument	argument type
at	*rtti::LineInfo* const implicit
name	string const implicit

Creates appropriate call expression for the given call function name in the *Program*. *ExprCallMacro* will be created if appropriate macro is found. Otherwise *ExprCall* will be created.

12.9 Visitor pattern

- *visit (program:smart_ptr<rtti::Program> const implicit;adapter:smart_ptr<ast::VisitorAdapter> const implicit;context:__context const;line:__lineInfo const) : void*
- *visit_modules (program:smart_ptr<rtti::Program> const implicit;adapter:smart_ptr<ast::VisitorAdapter> const implicit;context:__context const;line:__lineInfo const) : void*
- *visit (function:smart_ptr<ast::Function> const implicit;adapter:smart_ptr<ast::VisitorAdapter> const implicit;context:__context const;line:__lineInfo const) : void*
- *visit (expression:smart_ptr<ast::Expression> const implicit;adapter:smart_ptr<ast::VisitorAdapter> const implicit;context:__context const;line:__lineInfo const) : smart_ptr<ast::Expression>*
- *visit_finally (expression:smart_ptr<ast::ExprBlock> const implicit;adapter:smart_ptr<ast::VisitorAdapter> const implicit;context:__context const;line:__lineInfo const) : void*

visit(*program: smart_ptr<rtti::Program> const implicit; adapter: smart_ptr<ast::VisitorAdapter> const implicit*)

argument	argument type
program	smart_ptr< *rtti::Program* > const implicit
adapter	smart_ptr< *ast::VisitorAdapter* > const implicit

Invokes visitor for the given object.

visit_modules (*program: smart_ptr<rtti::Program> const implicit; adapter: smart_ptr<ast::VisitorAdapter> const implicit*)

argument	argument type
program	smart_ptr< *rtti::Program* > const implicit
adapter	smart_ptr< *ast::VisitorAdapter* > const implicit

Invokes visitor for the given list of modules inside the *Program*.

visit (*function: smart_ptr<ast::Function> const implicit; adapter: smart_ptr<ast::VisitorAdapter> const implicit*)

argument	argument type
function	smart_ptr< *ast::Function* > const implicit
adapter	smart_ptr< *ast::VisitorAdapter* > const implicit

Invokes visitor for the given object.

visit (*expression: smart_ptr<ast::Expression> const implicit; adapter: smart_ptr<ast::VisitorAdapter> const implicit*)

visit returns smart_ptr< *ast::Expression* >

argument	argument type
expression	smart_ptr< *ast::Expression* > const implicit
adapter	smart_ptr< *ast::VisitorAdapter* > const implicit

Invokes visitor for the given object.

visit_finally (*expression: smart_ptr<ast::ExprBlock> const implicit; adapter: smart_ptr<ast::VisitorAdapter> const implicit*)

argument	argument type
expression	smart_ptr< *ast::ExprBlock* > const implicit
adapter	smart_ptr< *ast::VisitorAdapter* > const implicit

Calls visit on the *finally* section of the block.

12.10 Expression generation

- *force_generated (expression:smart_ptr<ast::Expression> const& implicit;value:bool const) : void*
- *get_expression_annotation (expr:ast::Expression? const implicit;context:__context const;line:__lineInfo const) : rtti::Annotation?*
- *make_type_info_structure (ctx:rtti::Context implicit;type:smart_ptr<ast::TypeDecl> const implicit;context:__context const;at:__lineInfo const) : rtti::TypeInfo?*

force_generated(*expression: smart_ptr<ast::Expression> const& implicit; value: bool const*)

argument	argument type
expression	smart_ptr< *ast::Expression* > const& implicit
value	bool const

Forces *generated* flag on subexrepssion.

get_expression_annotation(*expr: ast::Expression? const implicit*)

get_expression_annotation returns *rtti::Annotation* ?

argument	argument type
expr	*ast::Expression* ? const implicit

Get 'Annotation' for the 'ast::Expression' and its inherited types.

make_type_info_structure(*ctx: Context implicit; type: smart_ptr<ast::TypeDecl> const implicit*)

make_type_info_structure returns *rtti::TypeInfo* ?

argument	argument type
ctx	*rtti::Context* implicit
type	smart_ptr< *ast::TypeDecl* > const implicit

Returns new *TypeInfo* corresponding to the specific type.

12.11 Adapter generation

- *make_visitor (class:void? const implicit;info:rtti::StructInfo const? const implicit;context:__context const) : smart_ptr<ast::VisitorAdapter>*
- *make_function_annotation (name:string const implicit;class:void? const implicit;info:rtti::StructInfo const? const implicit;context:__context const) : smart_ptr<ast::FunctionAnnotation>*
- *make_block_annotation (name:string const implicit;class:void? const implicit;info:rtti::StructInfo const? const implicit;context:__context const) : smart_ptr<ast::FunctionAnnotation>*
- *make_structure_annotation (name:string const implicit;class:void? const implicit;info:rtti::StructInfo const? const implicit;context:__context const) : smart_ptr<ast::StructureAnnotation>*
- *make_enumeration_annotation (name:string const implicit;class:void? const implicit;info:rtti::StructInfo const? const implicit;context:__context const) : smart_ptr<ast::EnumerationAnnotation>*
- *make_pass_macro (name:string const implicit;class:void? const implicit;info:rtti::StructInfo const? const implicit;context:__context const) : smart_ptr<ast::PassMacro>*
- *make_reader_macro (name:string const implicit;class:void? const implicit;info:rtti::StructInfo const? const implicit;context:__context const) : smart_ptr<ast::ReaderMacro>*
- *make_comment_reader (class:void? const implicit;info:rtti::StructInfo const? const implicit;context:__context const) : smart_ptr<ast::CommentReader>*
- *make_call_macro (name:string const implicit;class:void? const implicit;info:rtti::StructInfo const? const implicit;context:__context const) : smart_ptr<ast::CallMacro>*
- *make_typeinfo_macro (name:string const implicit;class:void? const implicit;info:rtti::StructInfo const? const implicit;context:__context const) : smart_ptr<ast::TypeInfoMacro>*
- *make_variant_macro (name:string const implicit;class:void? const implicit;info:rtti::StructInfo const? const implicit;context:__context const) : smart_ptr<ast::VariantMacro>*
- *make_for_loop_macro (name:string const implicit;class:void? const implicit;info:rtti::StructInfo const? const implicit;context:__context const) : smart_ptr<ast::ForLoopMacro>*
- *make_capture_macro (name:string const implicit;class:void? const implicit;info:rtti::StructInfo const? const implicit;context:__context const) : smart_ptr<ast::CaptureMacro>*
- *make_simulate_macro (name:string const implicit;class:void? const implicit;info:rtti::StructInfo const? const implicit;context:__context const) : smart_ptr<ast::SimulateMacro>*
- *make_clone_structure (structure:ast::Structure? const implicit) : smart_ptr<ast::Function>*
- *make_function_annotation (name:string const;someClassPtr:auto const) : smart_ptr<ast::FunctionAnnotation>*
- *make_block_annotation (name:string const;someClassPtr:auto const) : smart_ptr<ast::FunctionAnnotation>*
- *make_structure_annotation (name:string const;someClassPtr:auto const) : smart_ptr<ast::StructureAnnotation>*
- *make_enumeration_annotation (name:string const;someClassPtr:auto const) : smart_ptr<ast::EnumerationAnnotation>*
- *make_visitor (someClass:auto const) : smart_ptr<ast::VisitorAdapter>*
- *make_reader_macro (name:string const;someClassPtr:auto const) : smart_ptr<ast::ReaderMacro>*
- *make_comment_reader (name:string const;someClassPtr:auto const) : smart_ptr<ast::CommentReader>*
- *make_call_macro (name:string const;someClassPtr:auto const) : smart_ptr<ast::CallMacro>*

- *make_typeinfo_macro (name:string const;someClassPtr:auto const) : smart_ptr<ast::TypeInfoMacro>*
- *make_pass_macro (name:string const;someClassPtr:auto const) : smart_ptr<ast::PassMacro>*
- *make_variant_macro (name:string const;someClassPtr:auto const) : smart_ptr<ast::VariantMacro>*
- *make_for_loop_macro (name:string const;someClassPtr:auto const) : smart_ptr<ast::ForLoopMacro>*
- *make_capture_macro (name:string const;someClassPtr:auto const) : smart_ptr<ast::CaptureMacro>*
- *make_simulate_macro (name:string const;someClassPtr:auto const) : smart_ptr<ast::SimulateMacro>*

make_visitor (*class: void? const implicit; info: rtti::StructInfo const? const implicit*)

make_visitor returns smart_ptr< *ast::VisitorAdapter* >

argument	argument type
class	void? const implicit
info	*rtti::StructInfo* const? const implicit

Creates adapter for the *AstVisitor* interface.

make_function_annotation (*name: string const implicit; class: void? const implicit; info: rtti::StructInfo const? const implicit*)

make_function_annotation returns smart_ptr< *ast::FunctionAnnotation* >

argument	argument type
name	string const implicit
class	void? const implicit
info	*rtti::StructInfo* const? const implicit

Creates adapter for the *AstFunctionAnnotation*.

make_block_annotation (*name: string const implicit; class: void? const implicit; info: rtti::StructInfo const? const implicit*)

make_block_annotation returns smart_ptr< *ast::FunctionAnnotation* >

argument	argument type
name	string const implicit
class	void? const implicit
info	*rtti::StructInfo* const? const implicit

Creates adapter for the *AstBlockAnnotation*.

make_structure_annotation (*name: string const implicit; class: void? const implicit; info: rtti::StructInfo const? const implicit*) make_structure_annotation returns smart_ptr< *ast::StructureAnnotation* >

argument	argument type
name	string const implicit
class	void? const implicit
info	*rtti::StructInfo* const? const implicit

Creates adapter for the *AstStructureAnnotation*.

make_enumeration_annotation (*name: string const implicit; class: void? const implicit; info: rtti::StructInfo const? const implicit*)

make_enumeration_annotation returns smart_ptr< *ast::EnumerationAnnotation* >

argument	argument type
name	string const implicit
class	void? const implicit
info	*rtti::StructInfo* const? const implicit

Creates adapter for the *AstEnumearationAnnotation*.

make_pass_macro (*name: string const implicit; class: void? const implicit; info: rtti::StructInfo const? const implicit*)

make_pass_macro returns smart_ptr< *ast::PassMacro* >

argument	argument type
name	string const implicit
class	void? const implicit
info	*rtti::StructInfo* const? const implicit

Creates adapter for the *AstPassMacro*.

make_reader_macro (*name: string const implicit; class: void? const implicit; info: rtti::StructInfo const? const implicit*)

make_reader_macro returns smart_ptr< *ast::ReaderMacro* >

argument	argument type
name	string const implicit
class	void? const implicit
info	*rtti::StructInfo* const? const implicit

Creates adapter for the *AstReaderMacro*.

make_comment_reader (*class: void? const implicit; info: rtti::StructInfo const? const implicit*)

make_comment_reader returns smart_ptr< *ast::CommentReader* >

argument	argument type
class	void? const implicit
info	*rtti::StructInfo* const? const implicit

Creates adapter for the *AstCommentReader*.

make_call_macro (*name: string const implicit; class: void? const implicit; info: rtti::StructInfo const? const implicit*)

make_call_macro returns smart_ptr< *ast::CallMacro* >

argument	argument type
name	string const implicit
class	void? const implicit
info	*rtti::StructInfo* const? const implicit

Creates adapter for the *AstCallMacro*.

make_typeinfo_macro (*name: string const implicit; class: void? const implicit; info: rtti::StructInfo const? const implicit*)

make_typeinfo_macro returns smart_ptr< *ast::TypeInfoMacro* >

argument	argument type
name	string const implicit
class	void? const implicit
info	*rtti::StructInfo* const? const implicit

Creates adapter for the *AstTypeInfo* macro.

make_variant_macro (*name: string const implicit; class: void? const implicit; info: rtti::StructInfo const? const implicit*)

make_variant_macro returns smart_ptr< *ast::VariantMacro* >

argument	argument type
name	string const implicit
class	void? const implicit
info	*rtti::StructInfo* const? const implicit

Creates adapter for the *AstVariantMacro*.

make_for_loop_macro (*name: string const implicit; class: void? const implicit; info: rtti::StructInfo const? const implicit*)

make_for_loop_macro returns smart_ptr< *ast::ForLoopMacro* >

argument	argument type
name	string const implicit
class	void? const implicit
info	*rtti::StructInfo* const? const implicit

Creates adapter for the *AstForLoopMacro*.

make_capture_macro (*name: string const implicit; class: void? const implicit; info: rtti::StructInfo const? const implicit*)

make_capture_macro returns smart_ptr< *ast::CaptureMacro* >

argument	argument type
name	string const implicit
class	void? const implicit
info	*rtti::StructInfo* const? const implicit

Creates adapter for the *AstCaptureMacro*.

make_simulate_macro (*name: string const implicit; class: void? const implicit; info: rtti::StructInfo const? const implicit*)

make_simulate_macro returns smart_ptr< *ast::SimulateMacro* >

argument	argument type
name	string const implicit
class	void? const implicit
info	*rtti::StructInfo* const? const implicit

Creates adapter for the 'AstSimulateMacro' interface.

make_clone_structure (*structure: ast::Structure? const implicit*)

make_clone_structure returns smart_ptr< *ast::Function* >

argument	argument type
structure	*ast::Structure* ? const implicit

Generates *clone* function for the given structure.

make_function_annotation (*name: string const; someClassPtr: auto const*)

make_function_annotation returns *FunctionAnnotationPtr*

argument	argument type
name	string const
someClassPtr	auto const

Creates adapter for the *AstFunctionAnnotation*.

make_block_annotation (*name: string const; someClassPtr: auto const*)

make_block_annotation returns *FunctionAnnotationPtr*

argument	argument type
name	string const
someClassPtr	auto const

Creates adapter for the *AstBlockAnnotation*.

make_structure_annotation (*name: string const; someClassPtr: auto const*)

make_structure_annotation returns *StructureAnnotationPtr*

argument	argument type
name	string const
someClassPtr	auto const

Creates adapter for the *AstStructureAnnotation*.

make_enumeration_annotation (*name: string const; someClassPtr: auto const*)

make_enumeration_annotation returns *EnumerationAnnotationPtr*

argument	argument type
name	string const
someClassPtr	auto const

Creates adapter for the *AstEnumearationAnnotation*.

make_visitor (*someClass: auto const*)

make_visitor returns smart_ptr< *ast::VisitorAdapter* >

argument	argument type
someClass	auto const

Creates adapter for the *AstVisitor* interface.

make_reader_macro (*name: string const; someClassPtr: auto const*)

make_reader_macro returns *ReaderMacroPtr*

argument	argument type
name	string const
someClassPtr	auto const

Creates adapter for the *AstReaderMacro*.

make_comment_reader (*name: string const; someClassPtr: auto const*)

make_comment_reader returns *CommentReaderPtr*

argument	argument type
name	string const
someClassPtr	auto const

Creates adapter for the *AstCommentReader*.

make_call_macro (*name: string const; someClassPtr: auto const*)

make_call_macro returns *CallMacroPtr*

argument	argument type
name	string const
someClassPtr	auto const

Creates adapter for the *AstCallMacro*.

make_typeinfo_macro (*name: string const; someClassPtr: auto const*)

make_typeinfo_macro returns *TypeInfoMacroPtr*

argument	argument type
name	string const
someClassPtr	auto const

Creates adapter for the *AstTypeInfo* macro.

make_pass_macro (*name: string const; someClassPtr: auto const*)

make_pass_macro returns *PassMacroPtr*

argument	argument type
name	string const
someClassPtr	auto const

Creates adapter for the *AstPassMacro*.

make_variant_macro (*name: string const; someClassPtr: auto const*)

make_variant_macro returns *VariantMacroPtr*

argument	argument type
name	string const
someClassPtr	auto const

Creates adapter for the *AstVariantMacro*.

make_for_loop_macro (*name: string const; someClassPtr: auto const*)

make_for_loop_macro returns *ForLoopMacroPtr*

argument	argument type
name	string const
someClassPtr	auto const

Creates adapter for the *AstForLoopMacro*.

make_capture_macro (*name: string const; someClassPtr: auto const*)

make_capture_macro returns *CaptureMacroPtr*

argument	argument type
name	string const
someClassPtr	auto const

Creates adapter for the *AstCaptureMacro*.

make_simulate_macro (*name: string const; someClassPtr: auto const*)

make_simulate_macro returns *SimulateMacroPtr*

argument	argument type
name	string const
someClassPtr	auto const

Creates adapter for the 'AstSimulateMacro' interface.

12.12 Adapter application

- *add_function_annotation (module:rtti::Module? const implicit;annotation:smart_ptr<ast::FunctionAnnotation>& implicit;context:__context const;at:__lineInfo const) : void*
- *add_function_annotation (function:smart_ptr<ast::Function> const implicit;annotation:smart_ptr<ast::FunctionAnnotation>& implicit;context:__context const;at:__lineInfo const) : void*
- *add_function_annotation (function:smart_ptr<ast::Function> const implicit;annotation:smart_ptr<rtti::AnnotationDeclaration>& implicit;context:__context const;at:__lineInfo const) : void*
- *add_block_annotation (block:smart_ptr<ast::ExprBlock> const implicit;annotation:smart_ptr<ast::FunctionAnnotation>& implicit;context:__context const;at:__lineInfo const) : void*
- *add_block_annotation (block:smart_ptr<ast::ExprBlock> const implicit;annotation:smart_ptr<rtti::AnnotationDeclaration>& implicit;context:__context const;at:__lineInfo const) : void*
- *add_structure_annotation (module:rtti::Module? const implicit;annotation:smart_ptr<ast::StructureAnnotation>& implicit;context:__context const;at:__lineInfo const) : void*
- *add_structure_annotation (structure:smart_ptr<ast::Structure> const implicit;annotation:smart_ptr<ast::StructureAnnotation>& implicit;context:__context const;at:__lineInfo const) : void*
- *add_structure_annotation (structure:smart_ptr<ast::Structure> const implicit;annotation:smart_ptr<rtti::AnnotationDeclaration>& implicit;context:__context const;at:__lineInfo const) : void*
- *add_enumeration_annotation (module:rtti::Module? const implicit;annotation:smart_ptr<ast::EnumerationAnnotation>& implicit;context:__context const;at:__lineInfo const) : void*
- *add_infer_macro (module:rtti::Module? const implicit;annotation:smart_ptr<ast::PassMacro>& implicit;context:__context const) : void*
- *add_dirty_infer_macro (module:rtti::Module? const implicit;annotation:smart_ptr<ast::PassMacro>& implicit;context:__context const) : void*
- *add_lint_macro (module:rtti::Module? const implicit;annotation:smart_ptr<ast::PassMacro>& implicit;context:__context const) : void*
- *add_global_lint_macro (module:rtti::Module? const implicit;annotation:smart_ptr<ast::PassMacro>& implicit;context:__context const) : void*
- *add_optimization_macro (module:rtti::Module? const implicit;annotation:smart_ptr<ast::PassMacro>& implicit;context:__context const) : void*
- *add_reader_macro (module:rtti::Module? const implicit;annotation:smart_ptr<ast::ReaderMacro>& implicit;context:__context const;at:__lineInfo const) : void*
- *add_comment_reader (module:rtti::Module? const implicit;reader:smart_ptr<ast::CommentReader>& implicit;context:__context const;at:__lineInfo const) : void*
- *add_call_macro (module:rtti::Module? const implicit;annotation:smart_ptr<ast::CallMacro>& implicit;context:__context const;at:__lineInfo const) : void*
- *add_typeinfo_macro (module:rtti::Module? const implicit;annotation:smart_ptr<ast::TypeInfoMacro>& implicit;context:__context const;at:__lineInfo const) : void*
- *add_variant_macro (module:rtti::Module? const implicit;annotation:smart_ptr<ast::VariantMacro>& implicit;context:__context const) : void*
- *add_for_loop_macro (module:rtti::Module? const implicit;annotation:smart_ptr<ast::ForLoopMacro>& implicit;context:__context const) : void*

- *add_capture_macro (module:rtti::Module? const implicit;annotation:smart_ptr<ast::CaptureMacro>& implicit;context:__context const) : void*
- *add_simulate_macro (module:rtti::Module? const implicit;annotation:smart_ptr<ast::SimulateMacro>& implicit;context:__context const) : void*
- *add_module_option (module:rtti::Module? const implicit;option:string const implicit;type:rtti::Type const;context:__context const;at:__lineInfo const) : void*
- *add_new_block_annotation (name:string const;someClassPtr:auto const) : auto*
- *add_new_function_annotation (name:string const;someClassPtr:auto const) : auto*
- *add_new_contract_annotation (name:string const;someClassPtr:auto const) : auto*
- *add_new_structure_annotation (name:string const;someClassPtr:auto const) : auto*
- *add_new_enumeration_annotation (name:string const;someClassPtr:auto const) : auto*
- *add_new_variant_macro (name:string const;someClassPtr:auto const) : auto*
- *add_new_for_loop_macro (name:string const;someClassPtr:auto const) : auto*
- *add_new_capture_macro (name:string const;someClassPtr:auto const) : auto*
- *add_new_simulate_macro (name:string const;someClassPtr:auto const) : auto*
- *add_new_reader_macro (name:string const;someClassPtr:auto const) : auto*
- *add_new_comment_reader (name:string const;someClassPtr:auto const) : auto*
- *add_new_call_macro (name:string const;someClassPtr:auto const) : auto*
- *add_new_typeinfo_macro (name:string const;someClassPtr:auto const) : auto*
- *add_new_infer_macro (name:string const;someClassPtr:auto const) : auto*
- *add_new_dirty_infer_macro (name:string const;someClassPtr:auto const) : auto*
- *add_new_lint_macro (name:string const;someClassPtr:auto const) : auto*
- *add_new_global_lint_macro (name:string const;someClassPtr:auto const) : auto*
- *add_new_optimization_macro (name:string const;someClassPtr:auto const) : auto*

add_function_annotation (*module: rtti::Module? const implicit; annotation: smart_ptr<ast::FunctionAnnotation>& implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
annotation	smart_ptr< *ast::FunctionAnnotation* >& implicit

Adds function annotation to the given object. Calls *apply* if applicable.

add_function_annotation (*function: smart_ptr<ast::Function> const implicit; annotation: smart_ptr<ast::FunctionAnnotation>& implicit*)

argument	argument type
function	smart_ptr< *ast::Function* > const implicit
annotation	smart_ptr< *ast::FunctionAnnotation* >& implicit

Adds function annotation to the given object. Calls *apply* if applicable.

add_function_annotation (*function: smart_ptr<ast::Function> const implicit; annotation: smart_ptr<rtti::AnnotationDeclaration>& implicit*)

argument	argument type
function	smart_ptr< *ast::Function* > const implicit
annotation	smart_ptr< *rtti::AnnotationDeclaration* >& implicit

Adds function annotation to the given object. Calls *apply* if applicable.

add_block_annotation (*block: smart_ptr<ast::ExprBlock> const implicit; annotation: smart_ptr<ast::FunctionAnnotation>& implicit*)

argument	argument type
block	smart_ptr< *ast::ExprBlock* > const implicit
annotation	smart_ptr< *ast::FunctionAnnotation* >& implicit

Adds annotation declaration to the block.

add_block_annotation (*block: smart_ptr<ast::ExprBlock> const implicit; annotation: smart_ptr<rtti::AnnotationDeclaration>& implicit*)

argument	argument type
block	smart_ptr< *ast::ExprBlock* > const implicit
annotation	smart_ptr< *rtti::AnnotationDeclaration* >& implicit

Adds annotation declaration to the block.

add_structure_annotation (*module: rtti::Module? const implicit; annotation: smart_ptr<ast::StructureAnnotation>& implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
annotation	smart_ptr< *ast::StructureAnnotation* >& implicit

Adds structure annotation to the given object. Calls *apply* if applicable.

add_structure_annotation (*structure: smart_ptr<ast::Structure> const implicit; annotation: smart_ptr<ast::StructureAnnotation>& implicit*)

argument	argument type
structure	smart_ptr< *ast::Structure* > const implicit
annotation	smart_ptr< *ast::StructureAnnotation* >& implicit

Adds structure annotation to the given object. Calls *apply* if applicable.

add_structure_annotation(*structure: smart_ptr<ast::Structure> const implicit; annotation: smart_ptr<rtti::AnnotationDeclaration>& implicit*)

argument	argument type
structure	smart_ptr< *ast::Structure* > const implicit
annotation	smart_ptr< *rtti::AnnotationDeclaration* >& implicit

Adds structure annotation to the given object. Calls *apply* if applicable.

add_enumeration_annotation(*module: rtti::Module? const implicit; annotation: smart_ptr<ast::EnumerationAnnotation>& implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
annotation	smart_ptr< *ast::EnumerationAnnotation* >& implicit

Adds enumeration annotation to the given object. Calls *apply* if applicable.

add_infer_macro(*module: rtti::Module? const implicit; annotation: smart_ptr<ast::PassMacro>& implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
annotation	smart_ptr< *ast::PassMacro* >& implicit

Adds *AstPassMacro* adapter to the *infer* ` pass.

add_dirty_infer_macro(*module: rtti::Module? const implicit; annotation: smart_ptr<ast::PassMacro>& implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
annotation	smart_ptr< *ast::PassMacro* >& implicit

Adds *AstPassMacro* adapter to the *dirty infer* pass.

add_lint_macro (*module: rtti::Module? const implicit; annotation: smart_ptr<ast::PassMacro>& implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
annotation	smart_ptr< *ast::PassMacro* >& implicit

Adds *AstPassMacro* adapter to the *lint* pass.

add_global_lint_macro (*module: rtti::Module? const implicit; annotation: smart_ptr<ast::PassMacro>& implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
annotation	smart_ptr< *ast::PassMacro* >& implicit

Adds *AstPassMacro* adapter to the *global lint* pass.

add_optimization_macro (*module: rtti::Module? const implicit; annotation: smart_ptr<ast::PassMacro>& implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
annotation	smart_ptr< *ast::PassMacro* >& implicit

Adds *AstPassMacro* adapter to the *optimization* pass.

add_reader_macro (*module: rtti::Module? const implicit; annotation: smart_ptr<ast::ReaderMacro>& implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
annotation	smart_ptr< *ast::ReaderMacro* >& implicit

Adds *AstReaderMacro* adapter to the specific module.

add_comment_reader (*module: rtti::Module? const implicit; reader: smart_ptr<ast::CommentReader>& implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
reader	smart_ptr< *ast::CommentReader* >& implicit

Adds *AstCommentReader* adapter to the specific module.

add_call_macro (*module: rtti::Module? const implicit; annotation: smart_ptr<ast::CallMacro>& implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
annotation	smart_ptr< *ast::CallMacro* >& implicit

Adds *AstCallMacro* adapter to the specific module.

add_typeinfo_macro (*module: rtti::Module? const implicit; annotation: smart_ptr<ast::TypeInfoMacro>& implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
annotation	smart_ptr< *ast::TypeInfoMacro* >& implicit

Adds *AstTypeInfo* adapter to the specific module.

add_variant_macro (*module: rtti::Module? const implicit; annotation: smart_ptr<ast::VariantMacro>& implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
annotation	smart_ptr< *ast::VariantMacro* >& implicit

Adds *AstVariantMacro* to the specific module.

add_for_loop_macro (*module: rtti::Module? const implicit; annotation: smart_ptr<ast::ForLoopMacro>& implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
annotation	smart_ptr< *ast::ForLoopMacro* >& implicit

Adds *AstForLoopMacro* to the specific module.

add_capture_macro (*module: rtti::Module? const implicit; annotation: smart_ptr<ast::CaptureMacro>& implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
annotation	smart_ptr< *ast::CaptureMacro* >& implicit

Adds *AstCaptureMacro* to the specific module.

add_simulate_macro (*module: rtti::Module? const implicit; annotation: smart_ptr<ast::SimulateMacro>& implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
annotation	smart_ptr< *ast::SimulateMacro* >& implicit

Adds *AstSimulateMacro* to the specific module.

add_module_option (*module: rtti::Module? const implicit; option: string const implicit; type: Type const*)

argument	argument type
module	*rtti::Module* ? const implicit
option	string const implicit
type	*rtti::Type* const

Add module-specific option, which is accessible via "options" keyword.

add_new_block_annotation (*name: string const; someClassPtr: auto const*)

add_new_block_annotation returns auto

argument	argument type
name	string const
someClassPtr	auto const

Makes adapter to the *AstBlockAnnotation* and adds it to the current module.

add_new_function_annotation (*name: string const; someClassPtr: auto const*)

add_new_function_annotation returns auto

argument	argument type
name	string const
someClassPtr	auto const

Makes adapter to the *AstFunctionAnnotation* and adds it to the current module.

add_new_contract_annotation (*name: string const; someClassPtr: auto const*)

add_new_contract_annotation returns auto

argument	argument type
name	string const
someClassPtr	auto const

Makes adapter to the *AstContractAnnotation* and adds it to the current module.

add_new_structure_annotation (*name: string const; someClassPtr: auto const*)

add_new_structure_annotation returns auto

argument	argument type
name	string const
someClassPtr	auto const

Makes adapter to the *AstStructureAnnotation* and adds it to the current module.

add_new_enumeration_annotation (*name: string const; someClassPtr: auto const*)

add_new_enumeration_annotation returns auto

argument	argument type
name	string const
someClassPtr	auto const

Makes adapter to the *AstEnumerationAnnotation* and adds it to the current module.

add_new_variant_macro (*name: string const; someClassPtr: auto const*)

add_new_variant_macro returns auto

argument	argument type
name	string const
someClassPtr	auto const

Makes adapter to the *AstVariantMacro* and adds it to the current module.

add_new_for_loop_macro (*name: string const; someClassPtr: auto const*)

add_new_for_loop_macro returns auto

argument	argument type
name	string const
someClassPtr	auto const

Makes adapter to the *AstForLoopMacro* and adds it to the current module.

add_new_capture_macro (*name: string const; someClassPtr: auto const*)

add_new_capture_macro returns auto

argument	argument type
name	string const
someClassPtr	auto const

Makes adapter to the *AstCaptureMacro* and adds it to the current module.

add_new_simulate_macro (*name: string const; someClassPtr: auto const*)

add_new_simulate_macro returns auto

argument	argument type
name	string const
someClassPtr	auto const

Makes adapter to the *AstSimulateMacro* and adds it to the current module.

add_new_reader_macro (*name: string const; someClassPtr: auto const*)

add_new_reader_macro returns auto

argument	argument type
name	string const
someClassPtr	auto const

Makes adapter to the *AstReaderMacro* and adds it to the current module.

add_new_comment_reader (*name: string const; someClassPtr: auto const*)

add_new_comment_reader returns auto

argument	argument type
name	string const
someClassPtr	auto const

Makes adapter to the *AstCommentReader* and adds it to the current module.

add_new_call_macro (*name: string const; someClassPtr: auto const*)

add_new_call_macro returns auto

argument	argument type
name	string const
someClassPtr	auto const

Makes adapter to the *AstCallMacro* and adds it to the current module.

add_new_typeinfo_macro (*name: string const; someClassPtr: auto const*)

add_new_typeinfo_macro returns auto

argument	argument type
name	string const
someClassPtr	auto const

Makes adapter to the *AstTypeInfoMacro* and adds it to the current module.

add_new_infer_macro (*name: string const; someClassPtr: auto const*)

add_new_infer_macro returns auto

argument	argument type
name	string const
someClassPtr	auto const

Makes adapter to the *AstPassMacro* and adds it to the current module *infer* pass.

add_new_dirty_infer_macro (*name: string const; someClassPtr: auto const*)

add_new_dirty_infer_macro returns auto

argument	argument type
name	string const
someClassPtr	auto const

Makes adapter to the *AstPassMacro* and adds it to the current module *dirty infer* pass.

add_new_lint_macro (*name: string const; someClassPtr: auto const*)

add_new_lint_macro returns auto

argument	argument type
name	string const
someClassPtr	auto const

Makes adapter to the *AstPassMacro* and adds it to the current module *lint* pass.

add_new_global_lint_macro (*name: string const; someClassPtr: auto const*)

add_new_global_lint_macro returns auto

argument	argument type
name	string const
someClassPtr	auto const

Makes adapter to the *AstPassMacro* and adds it to the current module *global lint* pass.

add_new_optimization_macro (*name: string const; someClassPtr: auto const*)

add_new_optimization_macro returns auto

argument	argument type
name	string const
someClassPtr	auto const

Makes adapter to the *AstPassMacro* and adds it to the current module *optimization* pass.

12.13 Adding objects to objects

- *add_enumeration_entry (enum:smart_ptr<ast::Enumeration> const implicit;name:string const implicit) : int*
- *add_function (module:rtti::Module? const implicit;function:smart_ptr<ast::Function>& implicit;context:__context const;line:__lineInfo const) : bool*
- *add_generic (module:rtti::Module? const implicit;function:smart_ptr<ast::Function>& implicit;context:__context const;line:__lineInfo const) : bool*
- *add_variable (module:rtti::Module? const implicit;variable:smart_ptr<ast::Variable>& implicit;context:__context const;line:__lineInfo const) : bool*
- *add_keyword (module:rtti::Module? const implicit;keyword:string const implicit;needOxfordComma:bool const;context:__context const;line:__lineInfo const) : bool*
- *add_structure (module:rtti::Module? const implicit;structure:smart_ptr<ast::Structure>& implicit) : bool*
- *add_alias (module:rtti::Module? const implicit;structure:smart_ptr<ast::TypeDecl>& implicit) : bool*
- *add_module_require (module:rtti::Module? const implicit;publicModule:rtti::Module? const implicit;pub:bool const) : void*

add_enumeration_entry (*enum: smart_ptr<ast::Enumeration> const implicit; name: string const implicit*)

add_enumeration_entry returns int

argument	argument type
enum	smart_ptr< *ast::Enumeration* > const implicit
name	string const implicit

Adds entry to enumeration annotation.

add_function (*module: rtti::Module? const implicit; function: smart_ptr<ast::Function>& implicit*)

add_function returns bool

argument	argument type
module	*rtti::Module* ? const implicit
function	smart_ptr< *ast::Function* >& implicit

Adds function to a *Module*. Will return false on duplicates.

add_generic (*module: rtti::Module? const implicit; function: smart_ptr<ast::Function>& implicit*)

add_generic returns bool

argument	argument type
module	*rtti::Module* ? const implicit
function	smart_ptr< *ast::Function* >& implicit

Adds generic function to a *Module*. Will return false on duplicates.

add_variable (*module: rtti::Module? const implicit; variable: smart_ptr<ast::Variable>& implicit*)

add_variable returns bool

argument	argument type
module	*rtti::Module* ? const implicit
variable	smart_ptr< *ast::Variable* >& implicit

Adds variable to a *Module*. Will return false on duplicates.

add_keyword (*module: rtti::Module? const implicit; keyword: string const implicit; needOxfordComma: bool const*)

add_keyword returns bool

argument	argument type
module	*rtti::Module* ? const implicit
keyword	string const implicit
needOxfordComma	bool const

Adds new *keyword*. It can appear in the *keyword <type> expr* or *keyword expr block* syntax. See daslib/match as implementation example.

add_structure (*module: rtti::Module? const implicit; structure: smart_ptr<ast::Structure>& implicit*)

add_structure returns bool

argument	argument type
module	*rtti::Module* ? const implicit
structure	smart_ptr< *ast::Structure* >& implicit

Adds structure to a *Module*. Will return false on duplicates.

add_alias (*module: rtti::Module? const implicit; structure: smart_ptr<ast::TypeDecl>& implicit*)

add_alias returns bool

argument	argument type
module	*rtti::Module* ? const implicit
structure	smart_ptr< *ast::TypeDecl* >& implicit

Adds type alias to the specified module.

add_module_require (*module: rtti::Module? const implicit; publicModule: rtti::Module? const implicit; pub: bool const*)

argument	argument type
module	*rtti::Module* ? const implicit
publicModule	*rtti::Module* ? const implicit
pub	bool const

Add module dependencies similar to "require" keyword.

12.14 Program and module access

- *this_program (context:__context const) : smart_ptr<rtti::Program>*
- *this_module (context:__context const;line:__lineInfo const) : rtti::Module?*
- *compiling_program (context:__context const;at:__lineInfo const) : smart_ptr<rtti::Program>*
- *compiling_module (context:__context const;at:__lineInfo const) : rtti::Module?*

this_program ()

this_program returns smart_ptr< *rtti::Program* >

Program attached to the current context (or null if RTTI is disabled).

this_module ()

this_module returns *rtti::Module* ?

Main module attached to the current context (will through if RTTI is disabled).

compiling_program ()

compiling_program returns smart_ptr< *rtti::Program* >

Currently compiling program.

compiling_module ()

compiling_module returns *rtti::Module* ?

Currently compiling module.

12.15 Textual descriptions of the objects

- *describe_typedecl (type:smart_ptr<ast::TypeDecl> const implicit;extra:bool const;contracts:bool const;module:bool const;context:__context const;lineinfo:__lineInfo const) : string*
- *describe_typedecl_cpp (type:smart_ptr<ast::TypeDecl> const implicit;substitueRef:bool const;skipRef:bool const;skipConst:bool const;redundantConst:bool const;context:__context const;lineinfo:__lineInfo const) : string*
- *describe_expression (expression:smart_ptr<ast::Expression> const implicit;context:__context const;lineinfo:__lineInfo const) : string*
- *describe_function (function:smart_ptr<ast::Function> const implicit;context:__context const;lineinfo:__lineInfo const) : string*
- *das_to_string (type:rtti::Type const;context:__context const) : string*
- *describe (decl:smart_ptr<ast::TypeDecl> const;extra:bool const;contracts:bool const;modules:bool const) : auto*
- *describe_cpp (decl:smart_ptr<ast::TypeDecl> const;substitureRef:bool const;skipRef:bool const;skipConst:bool const;redundantConst:bool const) : auto*
- *describe (expr:smart_ptr<ast::Expression> const) : auto*
- *describe (expr:smart_ptr<ast::Function> const) : auto*

describe_typedecl (*type: smart_ptr<ast::TypeDecl> const implicit; extra: bool const; contracts: bool const; module: bool const*)

describe_typedecl returns string

argument	argument type
type	smart_ptr< *ast::TypeDecl* > const implicit
extra	bool const
contracts	bool const
module	bool const

Returns description of the *TypeDecl* which should match corresponding Daslang type declaration.

describe_typedecl_cpp (*type: smart_ptr<ast::TypeDecl> const implicit; substitueRef: bool const; skipRef: bool const; skipConst: bool const; redundantConst: bool const*)

describe_typedecl_cpp returns string

argument	argument type
type	smart_ptr< *ast::TypeDecl* > const implicit
substitueRef	bool const
skipRef	bool const
skipConst	bool const
redundantConst	bool const

Returns description of the *TypeDecl* which should match corresponding C++ type declaration.

describe_expression (*expression: smart_ptr<ast::Expression> const implicit*)

describe_expression returns string

argument	argument type
expression	smart_ptr< *ast::Expression* > const implicit

Returns description of the *Expression* which should match corresponding Daslang code.

describe_function (*function: smart_ptr<ast::Function> const implicit*)

describe_function returns string

argument	argument type
function	smart_ptr< *ast::Function* > const implicit

Returns description of the *Function* which should match corresponding Daslang function declaration.

das_to_string (*type: Type const*)

das_to_string returns string

argument	argument type
type	*rtti::Type* const

Returns description (name) of the corresponding *Type*.

describe (*decl: smart_ptr<ast::TypeDecl> const; extra: bool const; contracts: bool const; modules: bool const*)

describe returns auto

argument	argument type
decl	smart_ptr< *ast::TypeDecl* > const
extra	bool const
contracts	bool const
modules	bool const

Describes object and produces corresponding Daslang code as string.

describe_cpp (*decl: smart_ptr<ast::TypeDecl> const; substitureRef: bool const; skipRef: bool const; skipConst: bool const; redundantConst: bool const*)

describe_cpp returns auto

argument	argument type
decl	smart_ptr< *ast::TypeDecl* > const
substitureRef	bool const
skipRef	bool const
skipConst	bool const
redundantConst	bool const

Describes *TypeDecl* and produces corresponding C++ code as a string.

describe (*expr: smart_ptr<ast::Expression> const*)

describe returns auto

argument	argument type
expr	smart_ptr< *ast::Expression* > const

Describes object and produces corresponding Daslang code as string.

describe (*expr: smart_ptr<ast::Function> const*)

describe returns auto

argument	argument type
expr	smart_ptr< *ast::Function* > const

Describes object and produces corresponding Daslang code as string.

12.16 Searching

- *find_module_via_rtti (program:smart_ptr<rtti::Program> const implicit;name:string const implicit;context:__context const;lineinfo:__lineInfo const) : rtti::Module?*
- *find_module_function_via_rtti (module:rtti::Module? const implicit;function:function<> const;context:__context const;lineinfo:__lineInfo const) : smart_ptr<ast::Function>*
- *find_variable (module:rtti::Module? const implicit;variable:string const implicit) : smart_ptr<ast::Variable>*
- *find_bitfield_name (bit:smart_ptr<ast::TypeDecl> const implicit;value:bitfield const;context:__context const;lineinfo:__lineInfo const) : string*
- *find_enum_value (enum:smart_ptr<ast::Enumeration> const implicit;value:string const implicit) : int64*
- *find_structure_field (structPtr:ast::Structure? const implicit;field:string const implicit;context:__context const;lineinfo:__lineInfo const) : ast::FieldDeclaration?*
- *find_unique_structure (program:smart_ptr<rtti::Program> const implicit;name:string const implicit;context:__context const;at:__lineInfo const) : ast::Structure?*
- *find_module (prog:smart_ptr<rtti::Program> const;name:string const) : rtti::Module?*
- *find_module (name:string const) : rtti::Module?*
- *find_compiling_module (name:string const) : rtti::Module?*

find_module_via_rtti (*program: smart_ptr<rtti::Program> const implicit; name: string const implicit*)

find_module_via_rtti returns *rtti::Module* ?

argument	argument type
program	smart_ptr< *rtti::Program* > const implicit
name	string const implicit

Find module by name in the *Program*.

find_module_function_via_rtti (*module: rtti::Module? const implicit; function: function<> const*)

find_module_function_via_rtti returns smart_ptr< *ast::Function* >

argument	argument type
module	*rtti::Module* ? const implicit
function	function<> const

Find function by name in the *Module*.

find_variable (*module: rtti::Module? const implicit; variable: string const implicit*)

find_variable returns smart_ptr< *ast::Variable* >

argument	argument type
module	*rtti::Module* ? const implicit
variable	string const implicit

Finds variable in the *Module*.

find_bitfield_name (*bit: smart_ptr<ast::TypeDecl> const implicit; value: bitfield const*)

find_bitfield_name returns string

argument	argument type
bit	smart_ptr< *ast::TypeDecl* > const implicit
value	bitfield<> const

Finds name of the corresponding bitfield value in the specified type.

find_enum_value (*enum: smart_ptr<ast::Enumeration> const implicit; value: string const implicit*)

find_enum_value returns int64

argument	argument type
enum	smart_ptr< *ast::Enumeration* > const implicit
value	string const implicit

Finds name of the corresponding enumeration value in the specified type.

find_structure_field (*structPtr: ast::Structure? const implicit; field: string const implicit*)

find_structure_field returns *ast::FieldDeclaration* ?

argument	argument type
structPtr	*ast::Structure* ? const implicit
field	string const implicit

Returns *FieldDeclaration* for the specific field of the structure type, or *null* if not found.

find_unique_structure (*program: smart_ptr<rtti::Program> const implicit; name: string const implicit*)

find_unique_structure returns *ast::Structure* ?

argument	argument type
program	smart_ptr< *rtti::Program* > const implicit
name	string const implicit

Find structure in the program with the specified name. If its unique - return it, otherwise null.

find_module (*prog: smart_ptr<rtti::Program> const; name: string const*)

find_module returns *rtti::Module* ?

argument	argument type
prog	smart_ptr< *rtti::Program* > const
name	string const

Finds *Module* in the *Program*.

find_module (*name: string const*)

find_module returns *rtti::Module* ?

argument	argument type
name	string const

Finds *Module* in the *Program*.

find_compiling_module (*name: string const*)

find_compiling_module returns *rtti::Module* ?

argument	argument type
name	string const

Finds *Module* in the currently compiling *Program*.

12.17 Iterating

- *for_each_module (program:rtti::Program? const implicit;block:block<(var arg0:rtti::Module?):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *for_each_function (module:rtti::Module? const implicit;name:string const implicit;block:block<(var arg0:smart_ptr<ast::Function>):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *for_each_generic (module:rtti::Module? const implicit;name:string const implicit;block:block<(var arg0:smart_ptr<ast::Function>):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *any_table_foreach (table:void? const implicit;keyStride:int const;valueStride:int const;block:block<(var arg0:void?;var arg1:void?):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *any_array_foreach (array:void? const implicit;stride:int const;block:block<(var arg0:void?):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *for_each_typedef (module:rtti::Module? const implicit;block:block<(var arg0:string#;var arg1:smart_ptr<ast::TypeDecl>):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *for_each_enumeration (module:rtti::Module? const implicit;block:block<(var arg0:smart_ptr<ast::Enumeration>):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *for_each_structure (module:rtti::Module? const implicit;block:block<(var arg0:smart_ptr<ast::Structure>):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *for_each_generic (module:rtti::Module? const implicit;block:block<(var arg0:smart_ptr<ast::Function>):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *for_each_global (module:rtti::Module? const implicit;block:block<(var arg0:smart_ptr<ast::Variable>):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *for_each_call_macro (module:rtti::Module? const implicit;block:block<(var arg0:string#):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *for_each_reader_macro (module:rtti::Module? const implicit;block:block<(var arg0:string#):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *for_each_variant_macro (module:rtti::Module? const implicit;block:block<(var arg0:smart_ptr<ast::VariantMacro>):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *for_each_for_loop_macro (module:rtti::Module? const implicit;block:block<(var arg0:smart_ptr<ast::ForLoopMacro>):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *for_each_typeinfo_macro (module:rtti::Module? const implicit;block:block<(var arg0:smart_ptr<ast::TypeInfoMacro>):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *for_each_field (annotation:rtti::BasicStructureAnnotation const implicit;block:block<(var arg0:string;var arg1:string;var arg2:smart_ptr<ast::TypeDecl>;var arg3:uint):void> const implicit;context:__context const;line:__lineInfo const) : void*

for_each_module (*program: rtti::Program? const implicit; block: block<(var arg0:rtti::Module?):void> const implicit*)

argument	argument type
program	*rtti::Program* ? const implicit
block	block<(*rtti::Module* ?):void> const implicit

Iterates through each module in the program.

for_each_function (*module: rtti::Module? const implicit; name: string const implicit; block: block<(var arg0:smart_ptr<ast::Function>):void> const implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
name	string const implicit
block	block<(smart_ptr< *ast::Function* >):void> const implicit

Iterates through each function in the given *Module*. If the *name* is empty matches all functions.

for_each_generic (*module: rtti::Module? const implicit; name: string const implicit; block: block<(var arg0:smart_ptr<ast::Function>):void> const implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
name	string const implicit
block	block<(smart_ptr< *ast::Function* >):void> const implicit

Iterates through each generic function in the given *Module*.

any_table_foreach (*table: void? const implicit; keyStride: int const; valueStride: int const; block: block<(var arg0:void?;var arg1:void?):void> const implicit*)

argument	argument type
table	void? const implicit
keyStride	int const
valueStride	int const
block	block<(void?;void?):void> const implicit

Iterates through any table<> type in a typeless fasion (via void?)

any_array_foreach (*array: void? const implicit; stride: int const; block: block<(var arg0:void?):void> const implicit*)

argument	argument type
array	void? const implicit
stride	int const
block	block<(void?):void> const implicit

Iterates through any array<> type in a typeless fasion (via void?)

for_each_typedef (*module: rtti::Module? const implicit; block: block<(var arg0:string#;var arg1:smart_ptr<ast::TypeDecl>):void> const implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
block	block<(string#;smart_ptr< *ast::TypeDecl* >):void> const implicit

Iterates through every typedef in the *Module*.

for_each_enumeration (*module: rtti::Module? const implicit; block: block<(var arg0:smart_ptr<ast::Enumeration>):void> const implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
block	block<(smart_ptr< *ast::Enumeration* >):void> const implicit

Iterates through every enumeration in the *Module*.

for_each_structure (*module: rtti::Module? const implicit; block: block<(var arg0:smart_ptr<ast::Structure>):void> const implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
block	block<(smart_ptr< *ast::Structure* >):void> const implicit

Iterates through every structure in the *Module*.

for_each_generic (*module: rtti::Module? const implicit; block: block<(var arg0:smart_ptr<ast::Function>):void> const implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
block	block<(smart_ptr< *ast::Function* >):void> const implicit

Iterates through each generic function in the given *Module*.

for_each_global (*module: rtti::Module? const implicit; block: block<(var arg0:smart_ptr<ast::Variable>):void> const implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
block	block<(smart_ptr< *ast::Variable* >):void> const implicit

Iterates through every global variable in the *Module*.

for_each_call_macro (*module: rtti::Module? const implicit; block: block<(var arg0:string#):void> const implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
block	block<(string#):void> const implicit

Iterates through every CallMacro adapter in the *Module*.

for_each_reader_macro (*module: rtti::Module? const implicit; block: block<(var arg0:string#):void> const implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
block	block<(string#):void> const implicit

Iterates through each reader macro in the given *Module*.

for_each_variant_macro (*module: rtti::Module? const implicit; block: block<(var arg0:smart_ptr<ast::VariantMacro>):void> const implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
block	block<(smart_ptr< *ast::VariantMacro* >):void> const implicit

Iterates through each variant macro in the given *Module*.

for_each_for_loop_macro (*module: rtti::Module? const implicit; block: block<(var arg0:smart_ptr<ast::ForLoopMacro>):void> const implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
block	block<(smart_ptr< *ast::ForLoopMacro* >):void> const implicit

Iterates through each for loop macro in the given *Module*.

for_each_typeinfo_macro (*module: rtti::Module? const implicit; block: block<(var arg0:smart_ptr<ast::TypeInfoMacro>):void> const implicit*)

argument	argument type
module	*rtti::Module* ? const implicit
block	block<(smart_ptr< *ast::TypeInfoMacro* >):void> const implicit

Iterates through each typeinfo macro in the given *Module*.

for_each_field (*annotation: BasicStructureAnnotation const implicit; block: block<(var arg0:string;var arg1:string;var arg2:smart_ptr<ast::TypeDecl>;var arg3:uint):void> const implicit*)

argument	argument type
annotation	*rtti::BasicStructureAnnotation* const implicit
block	block<(string;string;smart_ptr< *ast::TypeDecl* >;uint):void> const implicit

Iterates through every field in the *BuiltinStructure* handled type.

12.18 Cloning

- *clone_structure (structure:ast::Structure const? const implicit) : smart_ptr<ast::Structure>*
- *clone_expression (expression:smart_ptr<ast::Expression> const implicit) : smart_ptr<ast::Expression>*
- *clone_function (function:smart_ptr<ast::Function> const implicit) : smart_ptr<ast::Function>*
- *clone_variable (variable:smart_ptr<ast::Variable> const implicit) : smart_ptr<ast::Variable>*
- *clone_type (type:smart_ptr<ast::TypeDecl> const implicit) : smart_ptr<ast::TypeDecl>*

clone_structure (*structure: ast::Structure const? const implicit*)

clone_structure returns smart_ptr< *ast::Structure* >

argument	argument type
structure	*ast::Structure* const? const implicit

Returns clone of the *Structure*.

clone_expression (*expression: smart_ptr<ast::Expression> const implicit*)

clone_expression returns smart_ptr< *ast::Expression* >

argument	argument type
expression	smart_ptr< *ast::Expression* > const implicit

Clones *Expression* with subexpressions, including corresponding type.

clone_function (*function: smart_ptr<ast::Function> const implicit*)

clone_function returns smart_ptr< *ast::Function* >

argument	argument type
function	smart_ptr< *ast::Function* > const implicit

Clones *Function* and everything in it.

clone_variable (*variable: smart_ptr<ast::Variable> const implicit*)

clone_variable returns smart_ptr< *ast::Variable* >

argument	argument type
variable	smart_ptr< *ast::Variable* > const implicit

Clones *Variable* and everything in it.

clone_type (*type: smart_ptr<ast::TypeDecl> const implicit*)

clone_type returns smart_ptr< *ast::TypeDecl* >

argument	argument type
type	smart_ptr< *ast::TypeDecl* > const implicit

Clones *TypeDecl* with subtypes.

12.19 Mangled name

- *parse_mangled_name (txt:string const implicit;lib:rtti::ModuleGroup implicit;thisModule:rtti::Module? const implicit;context:__context const;line:__lineInfo const) : smart_ptr<ast::TypeDecl>*
- *get_mangled_name (function:smart_ptr<ast::Function> const implicit;context:__context const;line:__lineInfo const) : string*
- *get_mangled_name (type:smart_ptr<ast::TypeDecl> const implicit;context:__context const;line:__lineInfo const) : string*

- *get_mangled_name (variable:smart_ptr<ast::Variable> const implicit;context:__context const;line:__lineInfo const) : string*
- *get_mangled_name (variable:smart_ptr<ast::ExprBlock> const implicit;context:__context const;line:__lineInfo const) : string*

parse_mangled_name (*txt: string const implicit; lib: ModuleGroup implicit; thisModule: rtti::Module? const implicit*)

parse_mangled_name returns smart_ptr< *ast::TypeDecl* >

argument	argument type
txt	string const implicit
lib	*rtti::ModuleGroup* implicit
thisModule	*rtti::Module* ? const implicit

Parses mangled name and creates corresponding *TypeDecl*.

get_mangled_name (*function: smart_ptr<ast::Function> const implicit*)

get_mangled_name returns string

argument	argument type
function	smart_ptr< *ast::Function* > const implicit

Returns mangled name of the object.

get_mangled_name (*type: smart_ptr<ast::TypeDecl> const implicit*)

get_mangled_name returns string

argument	argument type
type	smart_ptr< *ast::TypeDecl* > const implicit

Returns mangled name of the object.

get_mangled_name (*variable: smart_ptr<ast::Variable> const implicit*)

get_mangled_name returns string

argument	argument type
variable	smart_ptr< *ast::Variable* > const implicit

Returns mangled name of the object.

get_mangled_name (*variable: smart_ptr<ast::ExprBlock> const implicit*) get_mangled_name returns string

argument	argument type
variable	smart_ptr< *ast::ExprBlock* > const implicit

Returns mangled name of the object.

12.20 Size and offset

- *get_variant_field_offset (variant:smart_ptr<ast::TypeDecl> const implicit;index:int const;context:__context const;at:__lineInfo const) : int*
- *get_tuple_field_offset (typle:smart_ptr<ast::TypeDecl> const implicit;index:int const;context:__context const;at:__lineInfo const) : int*
- *any_array_size (array:void? const implicit) : int*
- *any_table_size (table:void? const implicit) : int*
- *get_handled_type_field_offset (type:smart_ptr<rtti::TypeAnnotation> const implicit;field:string const implicit;context:__context const;line:__lineInfo const) : uint*

get_variant_field_offset (*variant: smart_ptr<ast::TypeDecl> const implicit; index: int const*)

get_variant_field_offset returns int

argument	argument type
variant	smart_ptr< *ast::TypeDecl* > const implicit
index	int const

Returns offset of the variant field in bytes.

get_tuple_field_offset (*typle: smart_ptr<ast::TypeDecl> const implicit; index: int const*)

get_tuple_field_offset returns int

argument	argument type
typle	smart_ptr< *ast::TypeDecl* > const implicit
index	int const

Returns offset of the tuple field in bytes.

any_array_size (*array: void? const implicit*)

any_array_size returns int

argument	argument type
array	void? const implicit

Returns array size from pointer to array<> object.

any_table_size (*table: void? const implicit*)

any_table_size returns int

argument	argument type
table	void? const implicit

Returns table size from pointer to the table<> object.

get_handled_type_field_offset (*type: smart_ptr<rtti::TypeAnnotation> const implicit; field: string const implicit*)

get_handled_type_field_offset returns uint

argument	argument type
type	smart_ptr< *rtti::TypeAnnotation* > const implicit
field	string const implicit

Returns offset of the field in the ManagedStructure handled type.

12.21 Pointer conversion

- *ExpressionPtr (expr:smart_ptr<auto(TT)> const) : smart_ptr<ast::Expression>*
- *FunctionPtr (fun:ast::Function? const) : smart_ptr<ast::Function>*
- *StructurePtr (stru:ast::Structure? const) : smart_ptr<ast::Structure>*

ExpressionPtr (*expr: smart_ptr<auto(TT)> const*)

ExpressionPtr returns *ExpressionPtr*

argument	argument type
expr	smart_ptr<auto(TT)> const

Returns ExpressionPtr out of any smart pointer to *Expression*.

FunctionPtr (*fun: ast::Function? const*)

FunctionPtr returns *FunctionPtr*

argument	argument type
fun	*ast::Function* ? const

Returns FunctionPtr out of Function?

StructurePtr (*stru: ast::Structure? const*)

StructurePtr returns *StructurePtr*

argument	argument type
stru	*ast::Structure* ? const

Returns StructurePtr out of any smart pointer to *Structure*.

12.22 Evaluations

- *eval_single_expression (expr:smart_ptr<ast::Expression> const& implicit;ok:bool& implicit) : float4*

eval_single_expression (*expr: smart_ptr<ast::Expression> const& implicit; ok: bool& implicit*)

eval_single_expression returns float4

> **Warning:** This is unsafe operation.

argument	argument type
expr	smart_ptr< *ast::Expression* > const& implicit
ok	bool& implicit

Simulates and evaluates single expression on the separate context. If expression has external references, simulation will likely fail. Global variable access or function calls will produce exceptions.

12.23 Error reporting

- *macro_error (porogram:smart_ptr<rtti::Program> const implicit;at:rtti::LineInfo const implicit;message:string const implicit;context:__context const;line:__lineInfo const) : void*

macro_error (*porogram: smart_ptr<rtti::Program> const implicit; at: LineInfo const implicit; message: string const implicit*)

argument	argument type
porogram	smart_ptr< *rtti::Program* > const implicit
at	*rtti::LineInfo* const implicit
message	string const implicit

Reports error to the currently compiling program to whatever current pass is. Usually called from inside the macro function.

12.24 Location and context

- *force_at (expression:smart_ptr<ast::Expression> const& implicit;at:rtti::LineInfo const implicit) : void*
- *collect_dependencies (function:smart_ptr<ast::Function> const implicit;block:block<(var arg0:array<ast::Function?>;var arg1:array<ast::Variable?>):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *get_ast_context (program:smart_ptr<rtti::Program> const implicit;expression:smart_ptr<ast::Expression> const implicit;block:block<(var arg0:bool;var arg1:ast::AstContext):void> const implicit;context:__context const;line:__lineInfo const) : void*

force_at (*expression: smart_ptr<ast::Expression> const& implicit; at: LineInfo const implicit*)

argument	argument type
expression	smart_ptr< *ast::Expression* > const& implicit
at	*rtti::LineInfo* const implicit

Replaces line info in the expression, its subexpressions, and its types.

collect_dependencies (*function: smart_ptr<ast::Function> const implicit; block: block<(var arg0:array<ast::Function?>;var arg1:array<ast::Variable?>):void> const implicit*)

argument	argument type
function	smart_ptr< *ast::Function* > const implicit
block	block<(array< *ast::Function* ?>;array< *ast::Variable* ?>):void> const implicit

Collects dependencies of the given function (other functions it calls, global variables it accesses).

get_ast_context (*program: smart_ptr<rtti::Program> const implicit; expression: smart_ptr<ast::Expression> const implicit; block: block<(var arg0:bool;var arg1:ast::AstContext):void> const implicit*)

argument	argument type
program	smart_ptr< *rtti::Program* > const implicit
expression	smart_ptr< *ast::Expression* > const implicit
block	block<(bool; *ast::AstContext*):void> const implicit

Returns *AstContext* for the given expression. It includes current function (if applicable), loops, blocks, scopes, and with sections.

12.25 Use queries

- *get_use_global_variables (func:smart_ptr<ast::Function> const implicit;block:block<(var arg0:smart_ptr<ast::Variable>):void> const implicit;context:__context const;at:__lineInfo const) : void*
- *get_use_functions (func:smart_ptr<ast::Function> const implicit;block:block<(var arg0:smart_ptr<ast::Function>):void> const implicit;context:__context const;at:__lineInfo const) : void*

get_use_global_variables (*func: smart_ptr<ast::Function> const implicit; block: block<(var arg0:smart_ptr<ast::Variable>):void> const implicit*)

argument	argument type
func	smart_ptr< *ast::Function* > const implicit
block	block<(smart_ptr< *ast::Variable* >):void> const implicit

Provides invoked block with the list of all global variables, used by a function.

get_use_functions (*func: smart_ptr<ast::Function> const implicit; block: block<(var arg0:smart_ptr<ast::Function>):void> const implicit*)

argument	argument type
func	smart_ptr< *ast::Function* > const implicit
block	block<(smart_ptr< *ast::Function* >):void> const implicit

Provides invoked block with the list of all functions, used by a function.

12.26 Log

- *to_compilation_log (text:string const implicit;context:__context const;at:__lineInfo const) : void*

to_compilation_log(*text: string const implicit*)

argument	argument type
text	string const implicit

Writes to compilation log from macro during compilation.

12.27 Removal

- *remove_structure (module:rtti::Module? const implicit;structure:smart_ptr<ast::Structure>& implicit) : bool*

remove_structure(*module: rtti::Module? const implicit; structure: smart_ptr<ast::Structure>& implicit*)

remove_structure returns bool

argument	argument type
module	*rtti::Module* ? const implicit
structure	smart_ptr< *ast::Structure* >& implicit

Removes structure declaration from the specified module.

12.28 Properties

- *is_temp_type (type:smart_ptr<ast::TypeDecl> const implicit;refMatters:bool const) : bool*
- *is_same_type (leftType:smart_ptr<ast::TypeDecl> const implicit;rightType:smart_ptr<ast::TypeDecl> const implicit;refMatters:rtti::RefMatters const;constMatters:rtti::ConstMatters const;tempMatters:rtti::TemporaryMatters const;context:__context const;at:__lineInfo const) : bool*
- *get_underlying_value_type (type:smart_ptr<ast::TypeDecl> const implicit;context:__context const;line:__lineInfo const) : smart_ptr<ast::TypeDecl>*
- *get_handled_type_field_type (type:smart_ptr<rtti::TypeAnnotation> const implicit;field:string const implicit;context:__context const;line:__lineInfo const) : rtti::TypeInfo?*
- *has_field (type:smart_ptr<ast::TypeDecl> const implicit;fieldName:string const implicit;constant:bool const) : bool*
- *get_field_type (type:smart_ptr<ast::TypeDecl> const implicit;fieldName:string const implicit;constant:bool const) : smart_ptr<ast::TypeDecl>*
- *is_visible_directly (from_module:rtti::Module? const implicit;which_module:rtti::Module? const implicit) : bool*

- *is_expr_like_call (expression:smart_ptr<ast::Expression> const& implicit) : bool*
- *is_expr_const (expression:smart_ptr<ast::Expression> const& implicit) : bool*

is_temp_type (*type: smart_ptr<ast::TypeDecl> const implicit; refMatters: bool const*)

is_temp_type returns bool

argument	argument type
type	smart_ptr< *ast::TypeDecl* > const implicit
refMatters	bool const

Returns true if type can be temporary.

is_same_type (*leftType: smart_ptr<ast::TypeDecl> const implicit; rightType: smart_ptr<ast::TypeDecl> const implicit; refMatters: RefMatters const; constMatters: ConstMatters const; tempMatters: TemporaryMatters const*)

is_same_type returns bool

argument	argument type
leftType	smart_ptr< *ast::TypeDecl* > const implicit
rightType	smart_ptr< *ast::TypeDecl* > const implicit
refMatters	*rtti::RefMatters* const
constMatters	*rtti::ConstMatters* const
tempMatters	*rtti::TemporaryMatters* const

Compares two types given comparison parameters and returns true if they match.

get_underlying_value_type (*type: smart_ptr<ast::TypeDecl> const implicit*)

get_underlying_value_type returns smart_ptr< *ast::TypeDecl* >

argument	argument type
type	smart_ptr< *ast::TypeDecl* > const implicit

Returns Daslang type which is aliased with ManagedValue handled type.

get_handled_type_field_type (*type: smart_ptr<rtti::TypeAnnotation> const implicit; field: string const implicit*)

get_handled_type_field_type returns *rtti::TypeInfo* ?

argument	argument type
type	smart_ptr< *rtti::TypeAnnotation* > const implicit
field	string const implicit

Returns type of the field in the ManagedStructure handled type.

has_field (*type: smart_ptr<ast::TypeDecl> const implicit; fieldName: string const implicit; constant: bool const*)

has_field returns bool

argument	argument type
type	smart_ptr< *ast::TypeDecl* > const implicit
fieldName	string const implicit
constant	bool const

Returns if structure, variant, tuple, or handled type or pointer to either of those has specific field.

get_field_type (*type: smart_ptr<ast::TypeDecl> const implicit; fieldName: string const implicit; constant: bool const*)

get_field_type returns smart_ptr< *ast::TypeDecl* >

argument	argument type
type	smart_ptr< *ast::TypeDecl* > const implicit
fieldName	string const implicit
constant	bool const

Returns type of the field if structure, variant, tuple, or handled type or pointer to either of those has it. It's null otherwise.

is_visible_directly (*from_module: rtti::Module? const implicit; which_module: rtti::Module? const implicit*)

is_visible_directly returns bool

argument	argument type
from_module	*rtti::Module* ? const implicit
which_module	*rtti::Module* ? const implicit

Returns true if module is visible directly from the other module.

is_expr_like_call (*expression: smart_ptr<ast::Expression> const& implicit*)

is_expr_like_call returns bool

argument	argument type
expression	smart_ptr< *ast::Expression* > const& implicit

Returns true if expression is or inherited from *ExprLooksLikeCall*

is_expr_const (*expression: smart_ptr<ast::Expression> const& implicit*)

is_expr_const returns bool

argument	argument type
expression	smart_ptr< *ast::Expression* > const& implicit

Returns true if expression is or inherited from *ExprConst*

CHAPTER

THIRTEEN

BOOST PACKAGE FOR THE AST

The AST boost module implements collection of helper macros and functions to accompany *AST*.

All functions and symbols are in "ast_boost" module, use require to get access to it.

```
require daslib/ast_boost
```

13.1 Type aliases

AnnotationDeclarationPtr = smart_ptr<rtti::AnnotationDeclaration>

Alias for smart_ptr<AnnotationDeclaration>

DebugExpressionFlags is a bitfield

field	bit	value
refCount	0	1

Which things to print in debug_expression.

13.2 Function annotations

macro

MacroMacro function annotation.

tag_function

TagFunctionAnnotation function annotation.

13.3 Variant macros

better_rtti_in_expr

This macro is used to implement *is type*, *as type* and *?as type* runtime checks for the *Expression* class and its subclasses.

13.4 Structure macros

function_macro

Turns AstFunctionAnnotation into a macro with the specified *name*.

block_macro

Turns AstBlockAnnotation into a macro with the specified *name*.

structure_macro

Turns AstStructureAnnotation into a macro with the specified *name*.

enumeration_macro

Turns AstEnumerationAnnotation into a macro with the specified *name*.

contract

Turns AstFunctionAnnotation into a contract macro with the specified *name*.

reader_macro

Turns AstReaderMacro into a macro with the specified *name*.

comment_reader

Turns AstCommentReader into a macro with the specified *name*.

call_macro

Turns AstCallMacro into a macro with the specified *name*.

typeinfo_macro

Turns AstTypeInfoMacro into a macro with the specified *name*.

variant_macro

Turns AstVariantMacro into a macro with the specified *name*.

for_loop_macro

Turns AstForLoopMacro into a macro with the specified *name*.

capture_macro

Turns AstCaptureMacro into a macro with the specified *name*.

simulate_macro

Turns AstSimulateMacro into a macro with the specified *name*.

tag_structure

This macro implements [tag_structure] annotation, which allows to add tag (name) to a specific structure.

tag_function_macro

This macro implements [tag_function_macro] annotation, which allows to add an AstFunctionAnnotation to any function with a specific [tag_function(name)] tag.

infer_macro

Turns AstPassMacro into a macro with the specified 'name', which is called during the *infer* pass.

dirty_infer_macro

Turns AstPassMacro into a macro with the specified 'name', which is called during the *dirty infer* pass.

optimization_macro

Turns AstPassMacro into a macro with the specified 'name', which is called during the *optimization* pass.

lint_macro

Turns AstPassMacro into a macro with the specified 'name', which is called during the *lint* pass.

global_lint_macro

Turns AstPassMacro into a macro with the specified 'name', which is called during the *global lint* pass.

13.5 Classes

MacroMacro : AstFunctionAnnotation

This macro implements [macro] function annotation. This adds macro initialization function, which will only be called during macro compilation.

MacroMacro.**apply** (*self: AstFunctionAnnotation; func: FunctionPtr; group: ModuleGroup; args: AnnotationArgumentList const; errors: das_string*)

apply returns bool

argument	argument type
self	*ast::AstFunctionAnnotation*
func	*FunctionPtr*
group	*rtti::ModuleGroup*
args	*rtti::AnnotationArgumentList* const
errors	*builtin::das_string*

Implements [macro] function annotation. Internally it adds macro initialiation flag, as well as wraps function block in *if is_compiling_macros()* condition.

TagFunctionAnnotation : AstFunctionAnnotation

This annotation is used for tagging specific funcstion.

TagFunctionAnnotation.**apply** (*self: AstFunctionAnnotation; func: FunctionPtr; group: ModuleGroup; args: AnnotationArgumentList const; errors: das_string*)

apply returns bool

argument	argument type
self	*ast::AstFunctionAnnotation*
func	*FunctionPtr*
group	*rtti::ModuleGroup*
args	*rtti::AnnotationArgumentList* const
errors	*builtin::das_string*

Implements [tag_function] annotaiton. Internally this just verifies if tag has a name, i.e. bool argument without value (set to true).

TagStructureAnnotation : AstStructureAnnotation

This annotation is used for tagging specific structure. This annotation is used to tag structure with a name, which can be used to identify structure in the code.

TagStructureAnnotation.**apply** (*self: AstStructureAnnotation; st: StructurePtr; group: ModuleGroup; args: AnnotationArgumentList const; errors: das_string*)

apply returns bool

argument	argument type
self	*ast::AstStructureAnnotation*
st	*StructurePtr*
group	*rtti::ModuleGroup*
args	*rtti::AnnotationArgumentList* const
errors	*builtin::das_string*

Implements [tag_structure] annotation. Internally this just verifies if tag has a name, i.e. bool argument without value (set to true).

SetupAnyAnnotation : AstStructureAnnotation

This is base class for any annotation or macro setup.

it defines as follows

annotation_function_call : string
name : string

SetupAnyAnnotation.**apply** (*self: AstStructureAnnotation; st: StructurePtr; group: ModuleGroup; args: AnnotationArgumentList const; errors: das_string*)

apply returns bool

argument	argument type
self	*ast::AstStructureAnnotation*
st	*StructurePtr*
group	*rtti::ModuleGroup*
args	*rtti::AnnotationArgumentList* const
errors	*builtin::das_string*

Implements macro registration setup. Internally this creates __setup_macros function, which is only called during this module macro compilation. For the particular macro it adds call to the annotation registration function call (which is overrideable member *annotation_function_call*).

`SetupAnyAnnotation.`**`setup_call`**(*self: SetupAnyAnnotation; st: StructurePtr; cll: smart_ptr<ast::ExprCall>*)

argument	argument type
self	*ast_boost::SetupAnyAnnotation*
st	*StructurePtr*
cll	smart_ptr< *ast::ExprCall* >

Implements macro registration name setup. Internally this adds name parameter to the annotation registration function call (which is overridable member *name*).

`SetupFunctionAnnotation : SetupAnyAnnotation`

This is base class for function annotation setup.

it defines as follows

annotation_function_call : string
name : string

`SetupBlockAnnotation : SetupAnyAnnotation`

This is base class for block annotation setup.

it defines as follows

annotation_function_call : string
name : string

`SetupStructureAnnotation : SetupAnyAnnotation`

This is base class for structure annotation setup.

it defines as follows

annotation_function_call : string
name : string

SetupEnumerationAnnotation : SetupAnyAnnotation

[enumration_macro] implementation.

it defines as follows

annotation_function_call : string
name : string

SetupContractAnnotation : SetupAnyAnnotation

This is base class for contract annotation setup.

it defines as follows

annotation_function_call : string
name : string

SetupReaderMacro : SetupAnyAnnotation

[reader_macro] implementation.

it defines as follows

annotation_function_call : string
name : string

SetupCommentReader : SetupAnyAnnotation

[comment_reader] implementation.

it defines as follows

annotation_function_call : string
name : string

SetupVariantMacro : SetupAnyAnnotation

[variant_macro] implementation.

it defines as follows

annotation_function_call : string
name : string

SetupForLoopMacro : SetupAnyAnnotation

[for_loop_macro] implementation.

it defines as follows

annotation_function_call : string
name : string

SetupCaptureMacro : SetupAnyAnnotation

[capture_macro] implementation.

it defines as follows

annotation_function_call : string
name : string

SetupSimulateMacro : SetupAnyAnnotation

This is base class for a simulate macro. Internally this just verifies if tag has a name, i.e. bool argument without value (set to true).

it defines as follows

> annotation_function_call : string
> name : string

SetupCallMacro : SetupAnyAnnotation

[call_macro] implementation.

it defines as follows

> annotation_function_call : string
> name : string

SetupTypeInfoMacro : SetupAnyAnnotation

[typeinfo_macro] implementation.

it defines as follows

> annotation_function_call : string
> name : string

SetupInferMacro : SetupAnyAnnotation

[infer_macro] implementation.

it defines as follows

> annotation_function_call : string
> name : string

SetupDirtyInferMacro : SetupAnyAnnotation

[dirty_infer_macro] implementation.

it defines as follows

> annotation_function_call : string
> name : string

SetupLintMacro : SetupAnyAnnotation

[lint_macro] implementation.

it defines as follows

> annotation_function_call : string
> name : string

SetupGlobalLintMacro : SetupAnyAnnotation

[global_lint_macro] implementation.

it defines as follows

> annotation_function_call : string
> name : string

SetupOptimizationMacro : SetupAnyAnnotation

[optimization_macro] implementation.

it defines as follows

annotation_function_call : string
name : string

TagFunctionMacro : SetupAnyAnnotation

[tag_function_macro] implementation. Applies annotation to all tagged functions.

it defines as follows

annotation_function_call : string
name : string
tag : string

TagFunctionMacro.**apply** (*self: AstStructureAnnotation; st: StructurePtr; group: ModuleGroup; args: AnnotationArgumentList const; errors: das_string*)

apply returns bool

argument	argument type
self	*ast::AstStructureAnnotation*
st	*StructurePtr*
group	*rtti::ModuleGroup*
args	*rtti::AnnotationArgumentList* const
errors	*builtin::das_string*

Makes sure tag is defined and is a string.

TagFunctionMacro.**setup_call** (*self: SetupAnyAnnotation; st: StructurePtr; cll: smart_ptr<ast::ExprCall>*)

argument	argument type
self	*ast_boost::SetupAnyAnnotation*
st	*StructurePtr*
cll	smart_ptr< *ast::ExprCall* >

Attaches tag as well as name to the setup call.

BetterRttiVisitor : AstVariantMacro

Implements *expr is type* and *expr as type* checks, using RTTI.

BetterRttiVisitor.**visitExprIsVariant** (*self: AstVariantMacro; prog: ProgramPtr; mod: rtti::Module? const; expr: smart_ptr<ast::ExprIsVariant> const*)

visitExprIsVariant returns *ExpressionPtr*

argument	argument type
self	*ast::AstVariantMacro*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
expr	smart_ptr< *ast::ExprIsVariant* > const

Implements *is type*.

BetterRttiVisitor.**visitExprAsVariant** (*self: AstVariantMacro; prog: ProgramPtr; mod: rtti::Module? const; expr: smart_ptr<ast::ExprAsVariant> const*)

visitExprAsVariant returns *ExpressionPtr*

argument	argument type
self	*ast::AstVariantMacro*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
expr	smart_ptr< *ast::ExprAsVariant* > const

Implements *as type*.

BetterRttiVisitor.**visitExprSafeAsVariant** (*self: AstVariantMacro; prog: ProgramPtr; mod: rtti::Module? const; expr: smart_ptr<ast::ExprSafeAsVariant> const*)

visitExprSafeAsVariant returns *ExpressionPtr*

argument	argument type
self	*ast::AstVariantMacro*
prog	*ProgramPtr*
mod	*rtti::Module* ? const
expr	smart_ptr< *ast::ExprSafeAsVariant* > const

Implements *?as type*.

13.6 Containers

- *emplace_new (vec:$::dasvector`smart_ptr`Expression -const;ptr:smart_ptr<ast::Expression> -const) : void*
- *emplace_new (vec:$::dasvector`smart_ptr`TypeDecl -const;ptr:smart_ptr<ast::TypeDecl> -const) : void*
- *emplace_new (vec:$::dasvector`smart_ptr`Variable -const;ptr:smart_ptr<ast::Variable> -const) : void*
- *emplace_new (vec:ast::MakeStruct -const;ptr:smart_ptr<ast::MakeFieldDecl> -const) : void*

emplace_new (*vec: dasvector`smart_ptr`Expression; ptr: smart_ptr<ast::Expression>*)

argument	argument type
vec	vector<smart_ptr<Expression>>
ptr	smart_ptr< *ast::Expression* >

Emplaces newly created object into the container without memory leak (i.e. correct ptr_ref_count).

emplace_new (*vec: dasvector`smart_ptr`TypeDecl; ptr: smart_ptr<ast::TypeDecl>*)

argument	argument type
vec	vector<smart_ptr<TypeDecl>>
ptr	smart_ptr< *ast::TypeDecl* >

Emplaces newly created object into the container without memory leak (i.e. correct ptr_ref_count).

emplace_new (*vec: dasvector`smart_ptr`Variable; ptr: smart_ptr<ast::Variable>*)

argument	argument type
vec	vector<smart_ptr<Variable>>
ptr	smart_ptr< *ast::Variable* >

Emplaces newly created object into the container without memory leak (i.e. correct ptr_ref_count).

emplace_new (*vec: MakeStruct; ptr: smart_ptr<ast::MakeFieldDecl>*)

argument	argument type
vec	*ast::MakeStruct*
ptr	smart_ptr< *ast::MakeFieldDecl* >

Emplaces newly created object into the container without memory leak (i.e. correct ptr_ref_count).

13.7 Textual descriptions of the objects

- *describe (list:rtti::AnnotationArgumentList const) : string const*
- *describe (ann:rtti::AnnotationDeclaration const) : string*
- *describe (list:rtti::AnnotationList const) : string const*
- *describe (vvar:smart_ptr<ast::Variable> const) : string*
- *describe_function_short (func:smart_ptr<ast::Function> const) : string const*
- *debug_expression (expr:smart_ptr<ast::Expression> const;deFlags:bitfield<refCount> const) : string*
- *debug_expression (expr:ast::Expression? const) : string*
- *describe (expr:ast::Expression? const) : string*
- *describe_bitfield (bf:auto const;merger:string const) : auto*

describe (*list: AnnotationArgumentList const*)

describe returns string const

argument	argument type
list	*rtti::AnnotationArgumentList* const

Returns textual description of the object.

describe (*ann: AnnotationDeclaration const*)

describe returns string

argument	argument type
ann	*rtti::AnnotationDeclaration* const

Returns textual description of the object.

describe (*list: AnnotationList const*)

describe returns string const

argument	argument type
list	*rtti::AnnotationList* const

Returns textual description of the object.

describe (*vvar: VariablePtr*)

describe returns string

argument	argument type
vvar	*VariablePtr*

Returns textual description of the object.

describe_function_short (*func: FunctionPtr*)

describe_function_short returns string const

argument	argument type
func	*FunctionPtr*

Gives short (name, arguments with types, result type) description of the function.

debug_expression (*expr: ExpressionPtr; deFlags: DebugExpressionFlags*)

debug_expression returns string

argument	argument type
expr	*ExpressionPtr*
deFlags	*DebugExpressionFlags*

Gives hierarchical lisp-like textual representation of *expression* with all its subexpressions.

debug_expression (*expr: ast::Expression? const*)

debug_expression returns string

argument	argument type
expr	*ast::Expression* ? const

Gives hierarchical lisp-like textual representation of *expression* with all its subexpressions.

describe (*expr: ast::Expression? const*)

describe returns string

argument	argument type
expr	*ast::Expression* ? const

Returns textual description of the object.

describe_bitfield (*bf: auto const; merger: string const*) describe_bitfield returns auto

argument	argument type
bf	auto const
merger	string const

Returns textual description of the bitfield.

13.8 Queries

- *isVectorType (typ:rtti::Type const) : bool*
- *isExpression (t:smart_ptr<ast::TypeDecl> const;top:bool const) : bool*
- *is_same_or_inherited (parent:ast::Structure? const;child:ast::Structure? const) : bool const*
- *is_class_method (cinfo:smart_ptr<ast::Structure> const;finfo:smart_ptr<ast::TypeDecl> const) : bool const*
- *find_arg (argn:string const;args:rtti::AnnotationArgumentList const) : variant<tBool:bool;tInt:int;tUInt:uint;tInt64:int64;tUInt64:uint64;tFloat:float;tDouble:double;tString:string;nothing:any>*
- *find_arg (args:rtti::AnnotationArgumentList const;argn:string const) : variant<tBool:bool;tInt:int;tUInt:uint;tInt64:int64;tUInt64:uint64;tFloat:float;tDouble:double;tString:string;nothing:any>*
- *find_unique_function (mod:rtti::Module? const;name:string const;canfail:bool const) : smart_ptr<ast::Function>*
- *find_unique_generic (mod:rtti::Module? const;name:string const;canfail:bool const) : smart_ptr<ast::Function>*
- *find_annotation (mod_name:string const;ann_name:string const) : rtti::Annotation const?*
- *get_for_source_index (expr:smart_ptr<ast::ExprFor> const;svar:smart_ptr<ast::Variable> const) : int*
- *get_for_source_index (expr:smart_ptr<ast::ExprFor> const;source:smart_ptr<ast::Expression> const) : int*
- *isCMRES (fun:smart_ptr<ast::Function> const) : bool*
- *isCMRES (fun:ast::Function? const) : bool*
- *isMakeLocal (expr:smart_ptr<ast::Expression> const) : bool*
- *get_workhorse_types () : rtti::Type[30]*
- *find_argument_index (typ:smart_ptr<ast::TypeDecl> const;name:string const) : int*
- *isCMRESType (blockT:smart_ptr<ast::TypeDecl> const) : bool*
- *getVectorElementCount (bt:rtti::Type const) : int const*
- *getVectorElementSize (bt:rtti::Type const) : int const*
- *getVectorElementType (bt:rtti::Type const) : rtti::Type const*
- *getVectorOffset (bt:rtti::Type const;ident:string const) : int*

isVectorType (*typ: Type const*)

isVectorType returns bool

argument	argument type
typ	*rtti::Type* const

Returns true if type is vector type, i.e. int2, float3, and such, including range and urange.

isExpression (*t: TypeDeclPtr; top: bool const*)

isExpression returns bool

argument	argument type
t	*TypeDeclPtr*
top	bool const

Returns true if given object is derived from ast::Expression.

is_same_or_inherited (*parent: ast::Structure? const; child: ast::Structure? const*)

is_same_or_inherited returns bool const

argument	argument type
parent	*ast::Structure* ? const
child	*ast::Structure* ? const

Returns true if child is the same class as parent, or is inherited from the parent.

is_class_method (*cinfo: StructurePtr; finfo: TypeDeclPtr*)

is_class_method returns bool const

argument	argument type
cinfo	*StructurePtr*
finfo	*TypeDeclPtr*

Returns true if field is a class method.

find_arg (*argn: string const; args: AnnotationArgumentList const*)

find_arg returns *RttiValue*

> **Warning:** This function is deprecated.

argument	argument type
argn	string const
args	*rtti::AnnotationArgumentList* const

Find argument in annotation argument list.

find_arg (*args: AnnotationArgumentList const; argn: string const*)

find_arg returns *RttiValue*

argument	argument type
args	*rtti::AnnotationArgumentList* const
argn	string const

Find argument in annotation argument list.

find_unique_function (*mod: rtti::Module? const; name: string const; canfail: bool const*)

find_unique_function returns smart_ptr< *ast::Function* >

argument	argument type
mod	*rtti::Module* ? const
name	string const
canfail	bool const

Returns unique function of that specific name, or null if there is none or more than one.

find_unique_generic (*mod: rtti::Module? const; name: string const; canfail: bool const*)

find_unique_generic returns smart_ptr< *ast::Function* >

argument	argument type
mod	*rtti::Module* ? const
name	string const
canfail	bool const

Returns unique generic function of that specific name, or null if there is none or more than one.

find_annotation (*mod_name: string const; ann_name: string const*) find_annotation returns *rtti::Annotation* const?

argument	argument type
mod_name	string const
ann_name	string const

Finds annotation in the module.

get_for_source_index (*expr: smart_ptr<ast::ExprFor> const; svar: VariablePtr*)

get_for_source_index returns int

argument	argument type
expr	smart_ptr< *ast::ExprFor* > const
svar	*VariablePtr*

Find index of the for loop source variable.

get_for_source_index (*expr: smart_ptr<ast::ExprFor> const; source: ExpressionPtr*)

get_for_source_index returns int

argument	argument type
expr	smart_ptr< *ast::ExprFor* > const
source	*ExpressionPtr*

Find index of the for loop source variable.

isCMRES (*fun: FunctionPtr*)

isCMRES returns bool

argument	argument type
fun	*FunctionPtr*

Returns true if function returns result by copy-or-move on the stack, as oppose to through the register ABI.

isCMRES (*fun: ast::Function? const*)

isCMRES returns bool

argument	argument type
fun	*ast::Function* ? const

Returns true if function returns result by copy-or-move on the stack, as oppose to through the register ABI.

isMakeLocal (*expr: ExpressionPtr*)

isMakeLocal returns bool

argument	argument type
expr	*ExpressionPtr*

Returns true if Expression is inherited from ExprMakeLocal, i.e. ExprMakeArray, ExprMakeStruct, ExprMakeTuple, or ExprMakeVariant.

get_workhorse_types ()

get_workhorse_types returns *rtti::Type* [30]

Returns array which contains all *workhorse* base types.

find_argument_index (*typ: TypeDeclPtr; name: string const*)

find_argument_index returns int

argument	argument type
typ	*TypeDeclPtr*
name	string const

Returns index of the specific argument name, or -1 if its not found.

isCMRESType (*blockT: TypeDeclPtr*)

isCMRESType returns bool

argument	argument type
blockT	*TypeDeclPtr*

Returns true if type is copy-or-move on the stack, as oppose to through the register ABI.

getVectorElementCount (*bt: Type const*)

getVectorElementCount returns int const

argument	argument type
bt	*rtti::Type* const

Number of elements in the vector type, for example 3 for float3.

getVectorElementSize (*bt: Type const*) getVectorElementSize returns int const

argument	argument type
bt	*rtti::Type* const

Size of individual element in the vector type, for example 4 in float2 and 8 in range64.

getVectorElementType (*bt: Type const*)

getVectorElementType returns *rtti::Type* const

argument	argument type
bt	*rtti::Type* const

Type of individual element in the vector type, for example float in float2.

getVectorOffset (*bt: Type const; ident: string const*)

getVectorOffset returns int

argument	argument type
bt	*rtti::Type* const
ident	string const

Offset of the element in the vector type, for example 4 for “y” in float2.

13.9 Annotations

- *append_annotation (mod_name:string const;ann_name:string const;args:array<tuple<argname:string;argvalue:variant<tBool:b const) : smart_ptr<rtti::AnnotationDeclaration>*
- *append_annotation (mod_name:string const;ann_name:string const) : smart_ptr<rtti::AnnotationDeclaration>*
- *append_annotation (func:smart_ptr<ast::Function> -const;mod_name:string const;ann_name:string const) : void*
- *append_annotation (blk:smart_ptr<ast::ExprBlock> -const;mod_name:string const;ann_name:string const) : void*
- *append_annotation (st:smart_ptr<ast::Structure> -const;mod_name:string const;ann_name:string const) : void*
- *append_annotation (func:smart_ptr<ast::Function> -const;mod_name:string const;ann_name:string const;args:array<tuple<argname:string;argvalue:variant<tBool:bool;tInt:int;tUInt:uint;tInt64:int64;tUInt64:uint64;tFloat:floa const) : void*
- *append_annotation (blk:smart_ptr<ast::ExprBlock> -const;mod_name:string const;ann_name:string const;args:array<tuple<argname:string;argvalue:variant<tBool:bool;tInt:int;tUInt:uint;tInt64:int64;tUInt64:uint64;tFloat:floa const) : void*

- *append_annotation (st:smart_ptr<ast::Structure> -const;mod_name:string const;ann_name:string const;args:array<tuple<argname:string;argvalue:variant<tBool:bool;tInt:int;tUInt:uint;tInt64:int64;tUInt64:uint64;tFloat:floa const) : void*
- *add_annotation_argument (arguments:rtti::AnnotationArgumentList -const;argName:string const;val:bool const) : int const*
- *add_annotation_argument (arguments:rtti::AnnotationArgumentList -const;argName:string const;val:int const) : int const*
- *add_annotation_argument (arguments:rtti::AnnotationArgumentList -const;argName:string const;val:float const) : int const*
- *add_annotation_argument (arguments:rtti::AnnotationArgumentList -const;argName:string const;val:string const) : int const*
- *add_annotation_argument (arguments:rtti::AnnotationArgumentList -const;ann:rtti::AnnotationArgument const) : int const*

append_annotation (*mod_name: string const; ann_name: string const; args: array<tuple<argname:string;argvalue:variant<tBool:bool;tInt:int;tUInt:uint;tInt64:int64;tUInt64:uint64;tFl const*)

append_annotation returns smart_ptr< *rtti::AnnotationDeclaration* >

argument	argument type
mod_name	string const
ann_name	string const
args	array<tuple<argname:string;argvalue: *RttiValue* >> const

Appends function annotation to the function given its name and arguments.

append_annotation (*mod_name: string const; ann_name: string const*)

append_annotation returns smart_ptr< *rtti::AnnotationDeclaration* >

argument	argument type
mod_name	string const
ann_name	string const

Appends function annotation to the function given its name and arguments.

append_annotation (*func: FunctionPtr; mod_name: string const; ann_name: string const*)

argument	argument type
func	*FunctionPtr*
mod_name	string const
ann_name	string const

Appends function annotation to the function given its name and arguments.

append_annotation (*blk: smart_ptr<ast::ExprBlock>; mod_name: string const; ann_name: string const*)

argument	argument type
blk	smart_ptr< *ast::ExprBlock* >
mod_name	string const
ann_name	string const

Appends function annotation to the function given its name and arguments.

append_annotation (*st: smart_ptr<ast::Structure>; mod_name: string const; ann_name: string const*)

argument	argument type
st	smart_ptr< *ast::Structure* >
mod_name	string const
ann_name	string const

Appends function annotation to the function given its name and arguments.

append_annotation (*func: FunctionPtr; mod_name: string const; ann_name: string const; args: array<tuple<argname:string;argvalue:variant<tBool:bool;tInt:int;tUInt:uint;tInt64:int64;tUInt64:uint64;tFl const*)

argument	argument type
func	*FunctionPtr*
mod_name	string const
ann_name	string const
args	array<tuple<argname:string;argvalue: *RttiValue* >> const

Appends function annotation to the function given its name and arguments.

append_annotation (*blk: smart_ptr<ast::ExprBlock>; mod_name: string const; ann_name: string const; args: array<tuple<argname:string;argvalue:variant<tBool:bool;tInt:int;tUInt:uint;tInt64:int64;tUInt64:uint64;tFl const*)

argument	argument type
blk	smart_ptr< *ast::ExprBlock* >
mod_name	string const
ann_name	string const
args	array<tuple<argname:string;argvalue: *RttiValue* >> const

Appends function annotation to the function given its name and arguments.

append_annotation (*st: smart_ptr<ast::Structure>; mod_name: string const; ann_name: string const; args: array<tuple<argname:string;argvalue:variant<tBool:bool;tInt:int;tUInt:uint;tInt64:int64;tUInt64:uint64;tFl const*)

argument	argument type
st	smart_ptr< *ast::Structure* >
mod_name	string const
ann_name	string const
args	array<tuple<argname:string;argvalue: *RttiValue* >> const

Appends function annotation to the function given its name and arguments.

add_annotation_argument (*arguments: AnnotationArgumentList; argName: string const; val: bool const*)

add_annotation_argument returns int const

argument	argument type
arguments	*rtti::AnnotationArgumentList*
argName	string const
val	bool const

Adds annotation argument to the argument list.

add_annotation_argument (*arguments: AnnotationArgumentList; argName: string const; val: int const*)

add_annotation_argument returns int const

argument	argument type
arguments	*rtti::AnnotationArgumentList*
argName	string const
val	int const

Adds annotation argument to the argument list.

add_annotation_argument (*arguments: AnnotationArgumentList; argName: string const; val: float const*)

add_annotation_argument returns int const

argument	argument type
arguments	*rtti::AnnotationArgumentList*
argName	string const
val	float const

Adds annotation argument to the argument list.

add_annotation_argument (*arguments: AnnotationArgumentList; argName: string const; val: string const*)

add_annotation_argument returns int const

argument	argument type
arguments	*rtti::AnnotationArgumentList*
argName	string const
val	string const

Adds annotation argument to the argument list.

add_annotation_argument (*arguments: AnnotationArgumentList; ann: AnnotationArgument const*)

add_annotation_argument returns int const

argument	argument type
arguments	*rtti::AnnotationArgumentList*
ann	*rtti::AnnotationArgument* const

Adds annotation argument to the argument list.

13.10 Expression generation

- *override_method (str:smart_ptr<ast::Structure> -const;name:string const;funcName:string const) : bool*
- *panic_expr_as () : void?*
- *make_static_assert_false (text:string const;at:rtti::LineInfo const) : smart_ptr<ast::ExprStaticAssert>*
- *convert_to_expression (value:auto& ==const -const;at:rtti::LineInfo const) : auto*
- *convert_to_expression (value:auto const ==const;at:rtti::LineInfo const) : auto*
- *convert_to_expression (value:auto ==const -const) : auto*
- *convert_to_expression (value:auto const ==const) : auto*

override_method (*str: StructurePtr; name: string const; funcName: string const*)

override_method returns bool

argument	argument type
str	*StructurePtr*
name	string const
funcName	string const

Override class method *name* with new function.

panic_expr_as ()

panic_expr_as returns void?

Function call which panics with "invalid 'as' expression or null pointer dereference" message.

make_static_assert_false (*text: string const; at: LineInfo const*)

make_static_assert_false returns smart_ptr< *ast::ExprStaticAssert* >

argument	argument type
text	string const
at	*rtti::LineInfo* const

Creates *static_assert(false,text)* expression.

convert_to_expression (*value: auto& ==const; at: LineInfo const*)

convert_to_expression returns auto

argument	argument type
value	auto&!
at	*rtti::LineInfo* const

Converts value to expression, which generates this value.

convert_to_expression (*value: auto const ==const; at: LineInfo const*)

convert_to_expression returns auto

argument	argument type
value	auto const!
at	*rtti::LineInfo* const

Converts value to expression, which generates this value.

convert_to_expression (*value: auto ==const*)

convert_to_expression returns auto

argument	argument type
value	auto!

Converts value to expression, which generates this value.

convert_to_expression (*value: auto const ==const*)

convert_to_expression returns auto

argument	argument type
value	auto const!

Converts value to expression, which generates this value.

13.11 Visitors

- *visit_finally (blk:ast::ExprBlock? const;adapter:smart_ptr<ast::VisitorAdapter> const) : void*

visit_finally (*blk: ast::ExprBlock? const; adapter: smart_ptr<ast::VisitorAdapter> const*)

argument	argument type
blk	*ast::ExprBlock* ? const
adapter	smart_ptr< *ast::VisitorAdapter* > const

Calls visitor on the *finally* section of the block.

13.12 Type generation

- *function_to_type (fn:smart_ptr<ast::Function> const) : smart_ptr<ast::TypeDecl>*

function_to_type (*fn: FunctionPtr*)

function_to_type returns *TypeDeclPtr*

argument	argument type
fn	*FunctionPtr*

Returns TypeDeclPtr of the tFunction type, based on the provided function.

13.13 Setup

- *setup_call_list (name:string const;at:rtti::LineInfo const;subblock:block<(var fn:smart_ptr<ast::Function> -const):void> const) : ast::ExprBlock?*
- *setup_call_list (name:string const;at:rtti::LineInfo const;isInit:bool const;isPrivate:bool const;isLateInit:bool const) : ast::ExprBlock?*
- *setup_macro (name:string const;at:rtti::LineInfo const) : ast::ExprBlock?*
- *setup_tag_annotation (name:string const;tag:string const;classPtr:auto const) : auto*

setup_call_list (*name: string const; at: LineInfo const; subblock: block<(var fn:smart_ptr<ast::Function> -const):void> const*)

setup_call_list returns *ast::ExprBlock* ?

argument	argument type
name	string const
at	*rtti::LineInfo* const
subblock	block<(fn: *FunctionPtr*):void> const

Create new function which will contain collection of calls. Returns body block to where the call is to be appended.

setup_call_list (*name: string const; at: LineInfo const; isInit: bool const; isPrivate: bool const; isLateInit: bool const*)

setup_call_list returns *ast::ExprBlock* ?

argument	argument type
name	string const
at	*rtti::LineInfo* const
isInit	bool const
isPrivate	bool const
isLateInit	bool const

Create new function which will contain collection of calls. Returns body block to where the call is to be appended.

setup_macro (*name: string const; at: LineInfo const*)

setup_macro returns *ast::ExprBlock* ?

argument	argument type
name	string const
at	*rtti::LineInfo* const

Setup macro initialization function, which will only be called during compilation of this module. Returns body block to where the macro initialization is to be appended.

setup_tag_annotation (*name: string const; tag: string const; classPtr: auto const*)

setup_tag_annotation returns auto

argument	argument type
name	string const
tag	string const
classPtr	auto const

Creates annotation and applies it to all tagged functions given tag.

CHAPTER

FOURTEEN

STRING MANIPULATION LIBRARY

The string library implements string formatting, conversion, searching, and modification routines.

All functions and symbols are in "strings" module, use require to get access to it.

```
require strings
```

14.1 Handled structures

StringBuilderWriter

Object representing a string builder. Its significantly faster to write data to the string builder and than convert it to a string, as oppose to using sequences of string concatenations.

14.2 Character set

- *is_char_in_set (Character:int const;Charset:uint const[8] implicit;Context:__context const;At:__lineInfo const) : bool*
- *set_total (Charset:uint const[8] implicit) : uint*
- *set_element (Character:int const;Charset:uint const[8] implicit) : int*

is_char_in_set (*Character: int const; Charset: uint const[8] implicit*)

is_char_in_set returns bool

argument	argument type
Character	int const
Charset	uint const[8] implicit

Returns true if character bit is set in the set (of 256 bits in uint32[8]).

set_total (*Charset: uint const[8] implicit*)

set_total returns uint

argument	argument type
Charset	uint const[8] implicit

Total number of elements in the character set.

set_element (*Character: int const; Charset: uint const[8] implicit*)

set_element returns int

argument	argument type
Character	int const
Charset	uint const[8] implicit

Gen character set element by element index (not character index).

14.3 Character groups

- *is_alpha (Character:int const) : bool*
- *is_new_line (Character:int const) : bool*
- *is_white_space (Character:int const) : bool*
- *is_number (Character:int const) : bool*

is_alpha (*Character: int const*)

is_alpha returns bool

argument	argument type
Character	int const

Returns true if character is [A-Za-z].

is_new_line (*Character: int const*)

is_new_line returns bool

argument	argument type
Character	int const

Returns true if character is 'n' or 'r'.

is_white_space (*Character: int const*) is_white_space returns bool

argument	argument type
Character	int const

Returns true if character is [tnr].

is_number (*Character: int const*)

is_number returns bool

argument	argument type
Character	int const

Returns true if character is [0-9].

14.4 Character by index

- *character_at (str:string const implicit;idx:int const;context:__context const;at:__lineInfo const) : int*
- *character_uat (str:string const implicit;idx:int const) : int*

character_at (*str: string const implicit; idx: int const*)

character_at returns int

argument	argument type
str	string const implicit
idx	int const

Returns character of the string 'str' at index 'idx'.

character_uat (*str: string const implicit; idx: int const*)

character_uat returns int

Warning: This is unsafe operation.

argument	argument type
str	string const implicit
idx	int const

Returns character of the string 'str' at index 'idx'. This function does not check bounds of index.

14.5 String properties

- *ends_with (str:string const implicit;cmp:string const implicit;context:__context const) : bool*
- *ends_with (str:$::das_string const implicit;cmp:string const implicit;context:__context const) : bool*
- *starts_with (str:string const implicit;cmp:string const implicit;context:__context const) : bool*
- *starts_with (str:string const implicit;cmp:string const implicit;cmpLen:uint const;context:__context const) : bool*
- *starts_with (str:string const implicit;offset:int const;cmp:string const implicit;context:__context const) : bool*
- *starts_with (str:string const implicit;offset:int const;cmp:string const implicit;cmpLen:uint const;context:__context const) : bool*
- *length (str:string const implicit;context:__context const) : int*
- *length (str:$::das_string const implicit) : int*

ends_with (*str: string const implicit; cmp: string const implicit*)

ends_with returns bool

argument	argument type
str	string const implicit
cmp	string const implicit

returns *true* if the end of the string *str* matches a the string *cmp* otherwise returns *false*

ends_with (*str: das_string const implicit; cmp: string const implicit*)

ends_with returns bool

argument	argument type
str	*builtin::das_string* const implicit
cmp	string const implicit

returns *true* if the end of the string *str* matches a the string *cmp* otherwise returns *false*

starts_with (*str: string const implicit; cmp: string const implicit*)

starts_with returns bool

argument	argument type
str	string const implicit
cmp	string const implicit

returns *true* if the beginning of the string *str* matches the string *cmp*; otherwise returns *false*

starts_with (*str: string const implicit; cmp: string const implicit; cmpLen: uint const*)

starts_with returns bool

argument	argument type
str	string const implicit
cmp	string const implicit
cmpLen	uint const

returns *true* if the beginning of the string *str* matches the string *cmp*; otherwise returns *false*

starts_with (*str: string const implicit; offset: int const; cmp: string const implicit*)

starts_with returns bool

argument	argument type
str	string const implicit
offset	int const
cmp	string const implicit

returns *true* if the beginning of the string *str* matches the string *cmp*; otherwise returns *false*

starts_with (*str: string const implicit; offset: int const; cmp: string const implicit; cmpLen: uint const*)

starts_with returns bool

argument	argument type
str	string const implicit
offset	int const
cmp	string const implicit
cmpLen	uint const

returns *true* if the beginning of the string *str* matches the string *cmp*; otherwise returns *false*

length (*str: string const implicit*)

length returns int

argument	argument type
str	string const implicit

Return length of string

length (*str: das_string const implicit*)

length returns int

argument	argument type
str	*builtin::das_string* const implicit

Return length of string

14.6 String builder

- *build_string (block:block<(var arg0:strings::StringBuilderWriter):void> const implicit;context:__context const;lineinfo:__lineInfo const) : string*
- *write (writer:strings::StringBuilderWriter;anything:any) : strings::StringBuilderWriter&*
- *write_char (writer:strings::StringBuilderWriter implicit;ch:int const) : strings::StringBuilderWriter&*
- *write_chars (writer:strings::StringBuilderWriter implicit;ch:int const;count:int const) : strings::StringBuilderWriter&*
- *write_escape_string (writer:strings::StringBuilderWriter implicit;str:string const implicit) : strings::StringBuilderWriter&*
- *format (writer:strings::StringBuilderWriter implicit;format:string const implicit;value:int const) : strings::StringBuilderWriter&*
- *format (writer:strings::StringBuilderWriter implicit;format:string const implicit;value:uint const) : strings::StringBuilderWriter&*
- *format (writer:strings::StringBuilderWriter implicit;format:string const implicit;value:int64 const) : strings::StringBuilderWriter&*
- *format (writer:strings::StringBuilderWriter implicit;format:string const implicit;value:uint64 const) : strings::StringBuilderWriter&*
- *format (writer:strings::StringBuilderWriter implicit;format:string const implicit;value:float const) : strings::StringBuilderWriter&*
- *format (writer:strings::StringBuilderWriter implicit;format:string const implicit;value:double const) : strings::StringBuilderWriter&*
- *format (format:string const implicit;value:int const;context:__context const) : string*
- *format (format:string const implicit;value:uint const;context:__context const) : string*
- *format (format:string const implicit;value:int64 const;context:__context const) : string*
- *format (format:string const implicit;value:uint64 const;context:__context const) : string*

- *format (format:string const implicit;value:float const;context:__context const) : string*
- *format (format:string const implicit;value:double const;context:__context const) : string*

build_string (*block: block<(var arg0:strings::StringBuilderWriter):void> const implicit*)

build_string returns string

argument	argument type
block	block<(*strings::StringBuilderWriter*):void> const implicit

Create StringBuilderWriter and pass it to the block. Upon completion of a block, return whatever was written as string.

write (*writer: StringBuilderWriter; anything: any*)

write returns *strings::StringBuilderWriter* &

argument	argument type
writer	*strings::StringBuilderWriter*
anything	any

Returns textual representation of the value.

write_char (*writer: StringBuilderWriter implicit; ch: int const*)

write_char returns *strings::StringBuilderWriter* &

argument	argument type
writer	*strings::StringBuilderWriter* implicit
ch	int const

Writes character into StringBuilderWriter.

write_chars (*writer: StringBuilderWriter implicit; ch: int const; count: int const*)

write_chars returns *strings::StringBuilderWriter* &

argument	argument type
writer	*strings::StringBuilderWriter* implicit
ch	int const
count	int const

Writes multiple characters into StringBuilderWriter.

write_escape_string (*writer: StringBuilderWriter implicit; str: string const implicit*) write_escape_string returns *strings::StringBuilderWriter* &

argument	argument type
writer	*strings::StringBuilderWriter* implicit
str	string const implicit

Writes escaped string into StringBuilderWriter.

format (*writer: StringBuilderWriter implicit; format: string const implicit; value: int const*)

format returns *strings::StringBuilderWriter* &

argument	argument type
writer	*strings::StringBuilderWriter* implicit
format	string const implicit
value	int const

Converts value to string given specified format (that of C printf).

format (*writer: StringBuilderWriter implicit; format: string const implicit; value: uint const*)

format returns *strings::StringBuilderWriter* &

argument	argument type
writer	*strings::StringBuilderWriter* implicit
format	string const implicit
value	uint const

Converts value to string given specified format (that of C printf).

format (*writer: StringBuilderWriter implicit; format: string const implicit; value: int64 const*)

format returns *strings::StringBuilderWriter* &

argument	argument type
writer	*strings::StringBuilderWriter* implicit
format	string const implicit
value	int64 const

Converts value to string given specified format (that of C printf).

format (*writer: StringBuilderWriter implicit; format: string const implicit; value: uint64 const*)

format returns *strings::StringBuilderWriter* &

argument	argument type
writer	*strings::StringBuilderWriter* implicit
format	string const implicit
value	uint64 const

Converts value to string given specified format (that of C printf).

format (*writer: StringBuilderWriter implicit; format: string const implicit; value: float const*)

format returns *strings::StringBuilderWriter* &

argument	argument type
writer	*strings::StringBuilderWriter* implicit
format	string const implicit
value	float const

Converts value to string given specified format (that of C printf).

format (*writer: StringBuilderWriter implicit; format: string const implicit; value: double const*)

format returns *strings::StringBuilderWriter* &

argument	argument type
writer	*strings::StringBuilderWriter* implicit
format	string const implicit
value	double const

Converts value to string given specified format (that of C printf).

format (*format: string const implicit; value: int const*)

format returns string

argument	argument type
format	string const implicit
value	int const

Converts value to string given specified format (that of C printf).

format (*format: string const implicit; value: uint const*)

format returns string

argument	argument type
format	string const implicit
value	uint const

Converts value to string given specified format (that of C printf).

format (*format: string const implicit; value: int64 const*)

format returns string

argument	argument type
format	string const implicit
value	int64 const

Converts value to string given specified format (that of C printf).

format (*format: string const implicit; value: uint64 const*)

format returns string

argument	argument type
format	string const implicit
value	uint64 const

Converts value to string given specified format (that of C printf).

format (*format: string const implicit; value: float const*)

format returns string

argument	argument type
format	string const implicit
value	float const

Converts value to string given specified format (that of C printf).

format (*format: string const implicit; value: double const*) format returns string

argument	argument type
format	string const implicit
value	double const

Converts value to string given specified format (that of C printf).

14.7 das::string manipulation

- *append (str:$::das_string implicit;ch:int const) : void*
- *resize (str:$::das_string implicit;new_length:int const) : void*

append (*str: das_string implicit; ch: int const*)

argument	argument type
str	*builtin::das_string* implicit
ch	int const

Appends single character *ch* to das::string *str*.

resize (*str: das_string implicit; new_length: int const*)

argument	argument type
str	*builtin::das_string* implicit
new_length	int const

Resize string, i.e make it specified length.

14.8 String modifications

- *repeat (str:string const implicit;count:int const;context:__context const) : string*
- *strip (str:string const implicit;context:__context const) : string*
- *strip_right (str:string const implicit;context:__context const) : string*
- *strip_left (str:string const implicit;context:__context const) : string*
- *chop (str:string const implicit;start:int const;length:int const;context:__context const) : string*
- *slice (str:string const implicit;start:int const;end:int const;context:__context const) : string*

- *slice (str:string const implicit;start:int const;context:__context const) : string*
- *reverse (str:string const implicit;context:__context const) : string*
- *to_upper (str:string const implicit;context:__context const) : string*
- *to_lower (str:string const implicit;context:__context const) : string*
- *to_lower_in_place (str:string const implicit) : string*
- *to_upper_in_place (str:string const implicit) : string*
- *escape (str:string const implicit;context:__context const) : string*
- *unescape (str:string const implicit;context:__context const;at:__lineInfo const) : string*
- *safe_unescape (str:string const implicit;context:__context const) : string*
- *replace (str:string const implicit;toSearch:string const implicit;replace:string const implicit;context:__context const) : string*
- *rtrim (str:string const implicit;context:__context const) : string*
- *rtrim (str:string const implicit;chars:string const implicit;context:__context const) : string*

repeat (*str: string const implicit; count: int const*)

repeat returns string

argument	argument type
str	string const implicit
count	int const

Repeat string specified number of times, and return the result.

strip (*str: string const implicit*)

strip returns string

argument	argument type
str	string const implicit

Strips white-space-only characters that might appear at the beginning or end of the given string and returns the new stripped string.

strip_right (*str: string const implicit*)

strip_right returns string

argument	argument type
str	string const implicit

Strips white-space-only characters that might appear at the end of the given string and returns the new stripped string.

strip_left (*str: string const implicit*)

strip_left returns string

argument	argument type
str	string const implicit

Strips white-space-only characters that might appear at the beginning of the given string and returns the new stripped string.

chop (*str: string const implicit; start: int const; length: int const*)

chop returns string

argument	argument type
str	string const implicit
start	int const
length	int const

Return all part of the strings starting at start and ending at start + length.

slice (*str: string const implicit; start: int const; end: int const*)

slice returns string

argument	argument type
str	string const implicit
start	int const
end	int const

Return all part of the strings starting at start and ending by end. Start can be negative (-1 means "1 from the end").

slice (*str: string const implicit; start: int const*)

slice returns string

argument	argument type
str	string const implicit
start	int const

Return all part of the strings starting at start and ending by end. Start can be negative (-1 means "1 from the end").

reverse (*str: string const implicit*) reverse returns string

argument	argument type
str	string const implicit

Return reversed string

to_upper (*str: string const implicit*)

to_upper returns string

argument	argument type
str	string const implicit

Return all upper case string

to_lower (*str: string const implicit*)

to_lower returns string

argument	argument type
str	string const implicit

Return all lower case string

to_lower_in_place (*str: string const implicit*)

to_lower_in_place returns string

Warning: This is unsafe operation.

argument	argument type
str	string const implicit

Modify string in place to be all lower case

to_upper_in_place (*str: string const implicit*)

to_upper_in_place returns string

Warning: This is unsafe operation.

argument	argument type
str	string const implicit

Modify string in place to be all upper case string

escape (*str: string const implicit*)

escape returns string

argument	argument type
str	string const implicit

Escape string so that escape sequences are printable, for example converting "n" into "\n".

unescape (*str: string const implicit*)

unescape returns string

argument	argument type
str	string const implicit

Unescape string i.e reverse effects of *escape*. For example "\n" is converted to "n".

safe_unescape (*str: string const implicit*)

safe_unescape returns string

argument	argument type
str	string const implicit

Unescape string i.e reverse effects of *escape*. For example "\n" is converted to "n".

replace (*str: string const implicit; toSearch: string const implicit; replace: string const implicit*)

replace returns string

argument	argument type
str	string const implicit
toSearch	string const implicit
replace	string const implicit

Replace all occurances of the stubstring in the string with another substring.

rtrim (*str: string const implicit*)

rtrim returns string

argument	argument type
str	string const implicit

Removes trailing white space.

rtrim (*str: string const implicit; chars: string const implicit*)

rtrim returns string

argument	argument type
str	string const implicit
chars	string const implicit

Removes trailing white space.

14.9 Search substrings

- *find (str:string const implicit;substr:string const implicit;start:int const;context:__context const) : int*
- *find (str:string const implicit;substr:string const implicit) : int*
- *find (str:string const implicit;substr:int const;context:__context const) : int*
- *find (str:string const implicit;substr:int const;start:int const;context:__context const) : int*

find (*str: string const implicit; substr: string const implicit; start: int const*)

find returns int

argument	argument type
str	string const implicit
substr	string const implicit
start	int const

Return index where substr can be found within str (starting from optional 'start' at), or -1 if not found

find (*str: string const implicit; substr: string const implicit*)

find returns int

argument	argument type
str	string const implicit
substr	string const implicit

Return index where substr can be found within str (starting from optional 'start' at), or -1 if not found

find (*str: string const implicit; substr: int const*)

find returns int

argument	argument type
str	string const implicit
substr	int const

Return index where substr can be found within str (starting from optional 'start' at), or -1 if not found

find (*str: string const implicit; substr: int const; start: int const*)

find returns int

argument	argument type
str	string const implicit
substr	int const
start	int const

Return index where substr can be found within str (starting from optional 'start' at), or -1 if not found

14.10 String conversion routines

- *string (bytes:array<uint8> const implicit;context:__context const) : string*
- *to_char (char:int const;context:__context const) : string*
- *int (str:string const implicit;context:__context const;at:__lineInfo const) : int*
- *uint (str:string const implicit;context:__context const;at:__lineInfo const) : uint*
- *int64 (str:string const implicit;context:__context const;at:__lineInfo const) : int64*
- *uint64 (str:string const implicit;context:__context const;at:__lineInfo const) : uint64*
- *float (str:string const implicit;context:__context const;at:__lineInfo const) : float*
- *double (str:string const implicit;context:__context const;at:__lineInfo const) : double*
- *to_int (value:string const implicit;hex:bool const) : int*

- *to_uint (value:string const implicit;hex:bool const) : uint*
- *to_int64 (value:string const implicit;hex:bool const) : int64*
- *to_uint64 (value:string const implicit;hex:bool const) : uint64*
- *to_float (value:string const implicit) : float*
- *to_double (value:string const implicit) : double*

string (*bytes: array<uint8> const implicit*)

string returns string

argument	argument type
bytes	array<uint8> const implicit

Return string from the byte array.

to_char (*char: int const*)

to_char returns string

argument	argument type
char	int const

Convert character to string.

int (*str: string const implicit*)

int returns int

argument	argument type
str	string const implicit

Converts string to integer. In case of error panic.

uint (*str: string const implicit*)

uint returns uint

argument	argument type
str	string const implicit

Convert string to uint. In case of error panic.

int64 (*str: string const implicit*)

int64 returns int64

argument	argument type
str	string const implicit

Converts string to int64. In case of error panic.

uint64 (*str: string const implicit*)

uint64 returns uint64

argument	argument type
str	string const implicit

Convert string to uint64. In case of error panic.

float (*str: string const implicit*)

float returns float

argument	argument type
str	string const implicit

Converts string to float. In case of error panic.

double (*str: string const implicit*)

double returns double

argument	argument type
str	string const implicit

Converts string to double. In case of error panic.

to_int (*value: string const implicit; hex: bool const*)

to_int returns int

argument	argument type
value	string const implicit
hex	bool const

Convert string to int. In case of error returns 0

to_uint (*value: string const implicit; hex: bool const*) to_uint returns uint

argument	argument type
value	string const implicit
hex	bool const

Convert string to uint. In case of error returns 0u

to_int64 (*value: string const implicit; hex: bool const*)

to_int64 returns int64

argument	argument type
value	string const implicit
hex	bool const

Convert string to int64. In case of error returns 0l

to_uint64 (*value: string const implicit; hex: bool const*)

to_uint64 returns uint64

argument	argument type
value	string const implicit
hex	bool const

Convert string to uint64. In case of error returns 0ul

to_float (*value: string const implicit*)

to_float returns float

argument	argument type
value	string const implicit

Convert string to float. In case of error returns 0.0

to_double (*value: string const implicit*)

to_double returns double

argument	argument type
value	string const implicit

Convert string to double. In case of error returns 0.0lf

14.11 String as array

- *peek_data (str:string const implicit;block:block<(arg0:array<uint8> const#):void> const implicit;context:__context const;lineinfo:__lineInfo const) : void*
- *modify_data (str:string const implicit;block:block<(var arg0:array<uint8>#):void> const implicit;context:__context const;lineinfo:__lineInfo const) : string*

peek_data (*str: string const implicit; block: block<(arg0:array<uint8> const#):void> const implicit*)

argument	argument type
str	string const implicit
block	block<(array<uint8> const#):void> const implicit

Passes temporary array which is mapped to the string data to a block as read-only.

modify_data (*str: string const implicit; block: block<(var arg0:array<uint8>#):void> const implicit*)

modify_data returns string

argument	argument type
str	string const implicit
block	block<(array<uint8>#):void> const implicit

Passes temporary array which is mapped to the string data to a block for both reading and writing.

14.12 Low level memory allocation

- *delete_string (str:string& implicit;context:__context const) : void*
- *reserve_string_buffer (str:string const implicit;length:int const;context:__context const) : string*

delete_string (*str: string& implicit*)

> **Warning:** This is unsafe operation.

argument	argument type
str	string& implicit

Removes string from the string heap. This is unsafe because it will free the memory and all dangling strings will be broken.

reserve_string_buffer (*str: string const implicit; length: int const*)

reserve_string_buffer returns string

argument	argument type
str	string const implicit
length	int const

Allocate copy of the string data on the heap.

CHAPTER FIFTEEN

BOOST PACKAGE FOR STRING MANIPULATION LIBRARY

The STRINGS boost module implements collection of helper macros and functions to accompany *STRINGS*.

All functions and symbols are in "strings_boost" module, use require to get access to it.

```
require daslib/strings_boost
```

15.1 Split and join

- *split (text:string const implicit;delim:string const implicit) : array<string>*
- *split_by_chars (text:string const implicit;delim:string const implicit) : array<string>*
- *join (it:auto const;separator:string const implicit) : auto*
- *join (iterable:array<auto(TT)> const;separator:string const;blk:block<(var writer:strings::StringBuilderWriter -const;elem:TT const):void> const) : string*
- *join (iterable:iterator<auto(TT)> const;separator:string const;blk:block<(var writer:strings::StringBuilderWriter -const;elem:TT const):void> const) : string*
- *join (iterable:auto(TT) const[];separator:string const;blk:block<(var writer:strings::StringBuilderWriter -const;elem:TT const):void> const) : string*
- *split (text:string const implicit;delim:string const implicit;blk:block<(arg:array<string> const#):auto> const) : auto*
- *split_by_chars (text:string const implicit;delim:string const implicit;blk:block<(arg:array<string> const#):auto> const) : auto*

split (*text: string const implicit; delim: string const implicit*)

split returns array<string>

argument	argument type
text	string const implicit
delim	string const implicit

Split string given delimiter.

split_by_chars (*text: string const implicit; delim: string const implicit*) split_by_chars returns array<string>

argument	argument type
text	string const implicit
delim	string const implicit

Split string given set of delimiters (string treated as characters).

join (*it: auto const; separator: string const implicit*)

join returns auto

argument	argument type
it	auto const
separator	string const implicit

Join mulitiple strings with delimiter.

join (*iterable: array<auto(TT)> const; separator: string const; blk: block<(var writer:strings::StringBuilderWriter -const;elem:TT const):void> const*)

join returns string

argument	argument type
iterable	array<auto(TT)> const
separator	string const
blk	block<(writer: *strings::StringBuilderWriter* ;elem:TT const):void> const

Join mulitiple strings with delimiter.

join (*iterable: iterator<auto(TT)> const; separator: string const; blk: block<(var writer:strings::StringBuilderWriter -const;elem:TT const):void> const*)

join returns string

argument	argument type
iterable	iterator<auto(TT)> const
separator	string const
blk	block<(writer: *strings::StringBuilderWriter* ;elem:TT const):void> const

Join mulitiple strings with delimiter.

join (*iterable: auto(TT) const[ ]; separator: string const; blk: block<(var writer:strings::StringBuilderWriter -const;elem:TT const):void> const*)

join returns string

argument	argument type
iterable	auto(TT) const[-1]
separator	string const
blk	block<(writer: *strings::StringBuilderWriter* ;elem:TT const):void> const

Join mulitiple strings with delimiter.

split (*text: string const implicit; delim: string const implicit; blk: block<(arg:array<string> const#):auto> const*)

split returns auto

argument	argument type
text	string const implicit
delim	string const implicit
blk	block<(arg:array<string> const#):auto> const

Split string given delimiter.

split_by_chars (*text: string const implicit; delim: string const implicit; blk: block<(arg:array<string> const#):auto> const*)

split_by_chars returns auto

argument	argument type
text	string const implicit
delim	string const implicit
blk	block<(arg:array<string> const#):auto> const

Split string given set of delimiters (string treated as characters).

15.2 Formatting

- *wide (text:string const implicit;width:int const) : string*

wide (*text: string const implicit; width: int const*)

wide returns string

argument	argument type
text	string const implicit
width	int const

Pad string with ` ` character to make it certain width.

15.3 Queries and comparisons

- *is_character_at (foo:array<uint8> const implicit;idx:int const;ch:int const) : auto*
- *eq (a:string const implicit;b:$::das_string const) : auto*
- *eq (b:$::das_string const;a:string const implicit) : auto*

is_character_at (*foo: array<uint8> const implicit; idx: int const; ch: int const*)

is_character_at returns auto

argument	argument type
foo	array<uint8> const implicit
idx	int const
ch	int const

Returns true if specific character is at specific string position.

eq (*a: string const implicit; b: das_string const*)

eq returns auto

argument	argument type
a	string const implicit
b	*builtin::das_string* const

Compares das_string and string. True if equal.

eq (*b: das_string const; a: string const implicit*) eq returns auto

argument	argument type
b	*builtin::das_string* const
a	string const implicit

Compares das_string and string. True if equal.

15.4 Replace

- *replace_multiple (source:string const;replaces:array<tuple<text:string;replacement:string>> const) : string const*

replace_multiple (*source: string const; replaces: array<tuple<text:string;replacement:string>> const*)

replace_multiple returns string const

argument	argument type
source	string const
replaces	array<tuple<text:string;replacement:string>> const

Replace multiple substrings in string.

15.5 Levenshtein distance

- *levenshtein_distance (s:string const implicit;t:string const implicit) : int*
- *levenshtein_distance_fast (s:string const implicit;t:string const implicit) : int*

levenshtein_distance (*s: string const implicit; t: string const implicit*)

levenshtein_distance returns int

argument	argument type
s	string const implicit
t	string const implicit

Returns Levenshtein distance between two strings.

levenshtein_distance_fast (*s: string const implicit; t: string const implicit*) levenshtein_distance_fast returns int

argument	argument type
s	string const implicit
t	string const implicit

Returns Levenshtein distance between two strings, fast implementation.

15.6 Character traits

- *is_hex (ch:int const) : bool*
- *is_tab_or_space (ch:int const) : bool*

is_hex (*ch: int const*)

is_hex returns bool

argument	argument type
ch	int const

Returns true if character is hex digit.

is_tab_or_space (*ch: int const*)

is_tab_or_space returns bool

argument	argument type
ch	int const

Returns true if character is tab or space.

CHAPTER

SIXTEEN

FUNCTIONAL PROGRAMMING LIBRARY

The functional module implements a collection of high-order functions and patters to expose functional programming patters to Daslang.

All functions and symbols are in “functional” module, use require to get access to it.

```
require daslib/functional
```

16.1 Map, reduce

- *filter (src:iterator<auto(TT)> -const;blk:lambda<(what:TT const -&):bool> const) : auto*
- *filter (src:iterator<auto(TT)> -const;blk:function<(what:TT const -&):bool> const) : auto*
- *map (src:iterator<auto(TT)> -const;blk:lambda<(what:TT const -&):auto(QQ)> const) : auto*
- *map (src:iterator<auto(TT)> -const;blk:function<(what:TT const -&):auto(QQ)> const) : auto*
- *reduce (it:iterator<auto(TT)> const;blk:lambda<(left:TT const -&;right:TT const -&):TT const -&> const) : auto*
- *reduce (it:iterator<auto(TT)> const;blk:function<(left:TT const -&;right:TT const -&):TT const -&> const) : auto*
- *reduce (it:iterator<auto(TT)> const;blk:block<(left:TT const -&;right:TT const -&):TT const -&> const) : auto*
- *sum (it:iterator<auto(TT)> const) : auto*
- *any (it:auto const) : auto*
- *all (it:auto const) : auto*
- *cycle (src:iterator<auto(TT)> -const) : auto*
- *islice (src:iterator<auto(TT)> -const;start:int const;stop:int const) : auto*
- *repeat_ref (value:auto(TT) const;total:int -const) : auto*
- *repeat (value:auto(TT) const;count:int -const) : auto*
- *not (x:auto const) : auto*
- *echo (x:auto -const;extra:string const) : auto*
- *flatten (it:iterator<auto(TT)> -const) : auto*

filter (*src: iterator<auto(TT)>; blk: lambda<(what:TT const -&):bool> const*) filter returns auto

argument	argument type
src	iterator<auto(TT)>
blk	lambda<(what:TT const):bool> const

iterates over *src* and yields only those elements for which *blk* returns true

filter (*src: iterator<auto(TT)>; blk: function<(what:TT const -&):bool> const*)

filter returns auto

argument	argument type
src	iterator<auto(TT)>
blk	function<(what:TT const):bool> const

iterates over *src* and yields only those elements for which *blk* returns true

map (*src: iterator<auto(TT)>; blk: lambda<(what:TT const -&):auto(QQ)> const*)

map returns auto

argument	argument type
src	iterator<auto(TT)>
blk	lambda<(what:TT const):auto(QQ)> const

iterates over *src* and yields the result of *blk* for each element

map (*src: iterator<auto(TT)>; blk: function<(what:TT const -&):auto(QQ)> const*)

map returns auto

argument	argument type
src	iterator<auto(TT)>
blk	function<(what:TT const):auto(QQ)> const

iterates over *src* and yields the result of *blk* for each element

reduce (*it: iterator<auto(TT)> const; blk: lambda<(left:TT const -&;right:TT const -&):TT const -&> const*)

reduce returns auto

argument	argument type
it	iterator<auto(TT)> const
blk	lambda<(left:TT const;right:TT const):TT const> const

iterates over *it* and yields the reduced (combined) result of *blk* for each element and previous reduction result

reduce (*it: iterator<auto(TT)> const; blk: function<(left:TT const -&;right:TT const -&):TT const -&> const*)

reduce returns auto

argument	argument type
it	iterator<auto(TT)> const
blk	function<(left:TT const;right:TT const):TT const> const

iterates over *it* and yields the reduced (combined) result of *blk* for each element and previous reduction result

reduce (*it: iterator<auto(TT)> const; blk: block<(left:TT const -&;right:TT const -&):TT const -&> const*)

reduce returns auto

argument	argument type
it	iterator<auto(TT)> const
blk	block<(left:TT const;right:TT const):TT const> const

iterates over *it* and yields the reduced (combined) result of *blk* for each element and previous reduction result

sum (*it: iterator<auto(TT)> const*)

sum returns auto

argument	argument type
it	iterator<auto(TT)> const

iterates over *it* and yields the sum of all elements same as reduce(it, @(a,b) => a + b)

any (*it: auto const*)

any returns auto

argument	argument type
it	auto const

iterates over *it* and yields true if any element is true

all (*it: auto const*)

all returns auto

argument	argument type
it	auto const

iterates over *it* and yields true if all elements are true

cycle (*src: iterator<auto(TT)>*)

cycle returns auto

argument	argument type
src	iterator<auto(TT)>

endlessly iterates over *src*

islice (*src: iterator<auto(TT)>; start: int const; stop: int const*)

islice returns auto

argument	argument type
src	iterator<auto(TT)>
start	int const
stop	int const

iterates over *src* and yields only the elements in the range [start,stop)

repeat_ref (*value: auto(TT) const; total: int*)

repeat_ref returns auto

argument	argument type
value	auto(TT) const
total	int

yields *value* by reference *count* times

repeat (*value: auto(TT) const; count: int*)

repeat returns auto

argument	argument type
value	auto(TT) const
count	int

yields *value count* times

not (*x: auto const*)

not returns auto

argument	argument type
x	auto const

yeilds !x

echo (*x: auto; extra: string const*)

echo returns auto

argument	argument type
x	auto
extra	string const

prints contents of the string to the output, with *extra* string appended

flatten (*it: iterator<auto(TT)>*)

flatten returns auto

argument	argument type
it	iterator<auto(TT)>

iterates over *it*, than iterates over each element of each element of *it* and yields it

16.2 Queries

- *is_equal (a:auto const;b:auto const) : auto*
- *is_not_equal (a:auto const;b:auto const) : auto*

is_equal (*a: auto const; b: auto const*)

is_equal returns auto

argument	argument type
a	auto const
b	auto const

yields true if *a* and *b* are equal

is_not_equal (*a: auto const; b: auto const*)

is_not_equal returns auto

argument	argument type
a	auto const
b	auto const

yields true if *a* and *b* are not equal

16.3 Uncategorized

sorted (*arr: array<auto>*)

sorted returns auto

argument	argument type
arr	array<auto>

iterates over input and returns it sorted version

sorted (*it: iterator<auto(TT)>*)

sorted returns auto

argument	argument type
it	iterator<auto(TT)>

iterates over input and returns it sorted version

CHAPTER

SEVENTEEN

JOBS AND THREADS

Apply module implements job que and threading.

All functions and symbols are in "jobque" module, use require to get access to it.

```
require jobque
```

17.1 Handled structures

JobStatus

JobStatus property operators are

isReady	bool
isValid	bool
size	int

Job status indicator (ready or not, as well as entry count).

Channel

Channel property operators are

isEmpty	bool
total	int

Channel provides a way to communicate between multiple contexts, including threads and jobs. Channel has internal entry count.

LockBox

Lockbox. Similar to channel, only for single object.

Atomic32

Atomic 32 bit integer.

Atomic64

Atomic 64 bit integer.

17.2 Channel, JobStatus, Lockbox

- *lock_box_create (context:__context const;line:__lineInfo const) : jobque::LockBox?*
- *lock_box_remove (box:jobque::LockBox?& implicit;context:__context const;line:__lineInfo const) : void*
- *append (channel:jobque::JobStatus? const implicit;size:int const;context:__context const;line:__lineInfo const) : int*
- *channel_create (context:__context const;line:__lineInfo const) : jobque::Channel?*
- *channel_remove (channel:jobque::Channel?& implicit;context:__context const;line:__lineInfo const) : void*
- *add_ref (status:jobque::JobStatus? const implicit;context:__context const;line:__lineInfo const) : void*
- *release (status:jobque::JobStatus?& implicit;context:__context const;line:__lineInfo const) : void*
- *join (job:jobque::JobStatus? const implicit;context:__context const;line:__lineInfo const) : void*
- *notify (job:jobque::JobStatus? const implicit;context:__context const;line:__lineInfo const) : void*
- *notify_and_release (job:jobque::JobStatus?& implicit;context:__context const;line:__lineInfo const) : void*
- *job_status_create (context:__context const;line:__lineInfo const) : jobque::JobStatus?*
- *job_status_remove (jobStatus:jobque::JobStatus?& implicit;context:__context const;line:__lineInfo const) : void*

lock_box_create()

lock_box_create returns *jobque::LockBox* ?

Creates lockbox.

lock_box_remove(*box: jobque::LockBox?& implicit*)

Warning: This is unsafe operation.

argument	argument type
box	*jobque::LockBox* ?& implicit

Destroys lockbox.

append(*channel: jobque::JobStatus? const implicit; size: int const*)

append returns int

argument	argument type
channel	*jobque::JobStatus* ? const implicit
size	int const

Increase entry count to the channel.

channel_create ()

channel_create returns *jobque::Channel* ?

> **Warning:** This is unsafe operation.

Creates channel.

channel_remove (*channel: jobque::Channel?& implicit*)

> **Warning:** This is unsafe operation.

argument	argument type
channel	*jobque::Channel* ?& implicit

Destroys channel.

add_ref (*status: jobque::JobStatus? const implicit*)

argument	argument type
status	*jobque::JobStatus* ? const implicit

Increase reference count of the job status or channel.

release (*status: jobque::JobStatus?& implicit*)

argument	argument type
status	*jobque::JobStatus* ?& implicit

Decrease reference count of the job status or channel. Object is delete when reference count reaches 0.

join (*job: jobque::JobStatus? const implicit*)

argument	argument type
job	*jobque::JobStatus* ? const implicit

Wait until channel entry count reaches 0.

notify (*job: jobque::JobStatus? const implicit*)

argument	argument type
job	*jobque::JobStatus* ? const implicit

Notify channel that entry is completed (decrease entry count).

notify_and_release (*job: jobque::JobStatus?& implicit*)

argument	argument type
job	*jobque::JobStatus* ?& implicit

Notify channel or job status that entry is completed (decrease entry count) and decrease reference count of the job status or channel. Object is delete when reference count reaches 0.

job_status_create ()

job_status_create returns *jobque::JobStatus* ?

Creates job status.

job_status_remove (*jobStatus: jobque::JobStatus?& implicit*)

> **Warning:** This is unsafe operation.

argument	argument type
jobStatus	*jobque::JobStatus* ?& implicit

Destroys job status.

17.3 Queries

- *get_total_hw_jobs (context:__context const;line:__lineInfo const) : int*
- *get_total_hw_threads () : int*
- *is_job_que_shutting_down () : bool*

get_total_hw_jobs ()

get_total_hw_jobs returns int

Total number of hardware threads supporting job system.

get_total_hw_threads ()

get_total_hw_threads returns int

Total number of hardware threads available.

is_job_que_shutting_down ()

is_job_que_shutting_down returns bool

Returns true if job que infrastructure is shut-down or not initialized. This is useful for debug contexts, since it allows to check if job que is still alive.

17.4 Internal invocations

- *new_job_invoke (lambda:lambda<> const;function:function<> const;lambdaSize:int const;context:__context const;line:__lineInfo const) : void*
- *new_thread_invoke (lambda:lambda<> const;function:function<> const;lambdaSize:int const;context:__context const;line:__lineInfo const) : void*

new_job_invoke (*lambda: lambda<> const; function: function<> const; lambdaSize: int const*)

argument	argument type
lambda	lambda<> const
function	function<> const
lambdaSize	int const

Creates clone of the current context, moves attached lambda to it. Adds a job to a job que, which once invoked will execute the lambda on the context clone. *new_job_invoke* is part of the low level (internal) job infrastructure. Recommended approach is to use *jobque_boost::new_job*.

new_thread_invoke (*lambda: lambda<> const; function: function<> const; lambdaSize: int const*)

argument	argument type
lambda	lambda<> const
function	function<> const
lambdaSize	int const

Creates clone of the current context, moves attached lambda to it. Creates a thread, invokes the lambda on the new context in that thread. *new_thread_invoke* is part of the low level (internal) thread infrastructure. Recommended approach is to use *jobque_boost::new_thread*.

17.5 Construction

- *with_lock_box (block:block<(var arg0:jobque::LockBox?):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *with_channel (block:block<(var arg0:jobque::Channel?):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *with_channel (count:int const;block:block<(var arg0:jobque::Channel?):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *with_job_status (total:int const;block:block<(var arg0:jobque::JobStatus?):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *with_job_que (block:block<void> const implicit;context:__context const;line:__lineInfo const) : void*

with_lock_box (*block: block<(var arg0:jobque::LockBox?):void> const implicit*)

argument	argument type
block	block<(*jobque::LockBox* ?):void> const implicit

Creates *LockBox*, makes it available inside the scope of the block.

with_channel (*block: block<(var arg0:jobque::Channel?):void> const implicit*)

argument	argument type
block	block<(*jobque::Channel* ?):void> const implicit

Creates *Channel*, makes it available inside the scope of the block.

with_channel (*count: int const; block: block<(var arg0:jobque::Channel?):void> const implicit*)

argument	argument type
count	int const
block	block<(*jobque::Channel* ?):void> const implicit

Creates *Channel*, makes it available inside the scope of the block.

with_job_status (*total: int const; block: block<(var arg0:jobque::JobStatus?):void> const implicit*)

argument	argument type
total	int const
block	block<(*jobque::JobStatus* ?):void> const implicit

Creates *JobStatus*, makes it available inside the scope of the block.

with_job_que (*block: block<void> const implicit*)

argument	argument type
block	block<> const implicit

Makes sure jobque infrastructure is available inside the scope of the block. There is cost associated with creating such infrastructure (i.e. creating hardware threads, jobs, etc). If jobs are integral part of the application, with_job_que should be high in the call stack. If it`s a one-off - it should be encricled accordingly to reduce runtime memory footprint of the application.

17.6 Atomic

- *atomic32_create (context:__context const;line:__lineInfo const) : jobque::Atomic32?*
- *atomic32_remove (atomic:jobque::Atomic32?& implicit;context:__context const;line:__lineInfo const) : void*
- *with_atomic32 (block:block<(var arg0:jobque::Atomic32?):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *set (atomic:jobque::Atomic32? const implicit;value:int const;context:__context const;line:__lineInfo const) : void*
- *get (atomic:jobque::Atomic32? const implicit;context:__context const;line:__lineInfo const) : int*
- *inc (atomic:jobque::Atomic32? const implicit;context:__context const;line:__lineInfo const) : int*
- *dec (atomic:jobque::Atomic32? const implicit;context:__context const;line:__lineInfo const) : int*
- *atomic64_create (context:__context const;line:__lineInfo const) : jobque::Atomic64?*
- *atomic64_remove (atomic:jobque::Atomic64?& implicit;context:__context const;line:__lineInfo const) : void*
- *with_atomic64 (block:block<(var arg0:jobque::Atomic64?):void> const implicit;context:__context const;line:__lineInfo const) : void*
- *set (atomic:jobque::Atomic64? const implicit;value:int64 const;context:__context const;line:__lineInfo const) : void*
- *get (atomic:jobque::Atomic64? const implicit;context:__context const;line:__lineInfo const) : int64*
- *inc (atomic:jobque::Atomic64? const implicit;context:__context const;line:__lineInfo const) : int64*
- *dec (atomic:jobque::Atomic64? const implicit;context:__context const;line:__lineInfo const) : int64*

atomic32_create ()

atomic32_create returns *jobque::Atomic32* ?

Creates atomic 32 bit integer.

atomic32_remove (*atomic: jobque::Atomic32?& implicit*)

> **Warning:** This is unsafe operation.

argument	argument type
atomic	*jobque::Atomic32* ?& implicit

Destroys atomic 32 bit integer.

with_atomic32 (*block: block<(var arg0:jobque::Atomic32?):void> const implicit*)

argument	argument type
block	block<(*jobque::Atomic32* ?):void> const implicit

Creates *Atomic32*, makes it available inside the scope of the block.

set (*atomic: jobque::Atomic32? const implicit; value: int const*)

argument	argument type
atomic	*jobque::Atomic32* ? const implicit
value	int const

Set atomic integer value.

get (*atomic: jobque::Atomic32? const implicit*)

get returns int

argument	argument type
atomic	*jobque::Atomic32* ? const implicit

Get atomic integer value.

inc (*atomic: jobque::Atomic32? const implicit*)

inc returns int

argument	argument type
atomic	*jobque::Atomic32* ? const implicit

Increase atomic integer value and returns result.

dec (*atomic: jobque::Atomic32? const implicit*)

dec returns int

argument	argument type
atomic	*jobque::Atomic32* ? const implicit

Decrease atomic integer value and returns result.

atomic64_create ()

atomic64_create returns *jobque::Atomic64* ?

Creates atomic 64 bit integer.

atomic64_remove (*atomic: jobque::Atomic64?& implicit*)

> **Warning:** This is unsafe operation.

argument	argument type
atomic	*jobque::Atomic64* ?& implicit

Destroys atomic 64 bit integer.

with_atomic64 (*block: block<(var arg0:jobque::Atomic64?):void> const implicit*)

argument	argument type
block	block<(*jobque::Atomic64* ?):void> const implicit

Creates *Atomic64*, makes it available inside the scope of the block.

set (*atomic: jobque::Atomic64? const implicit; value: int64 const*)

argument	argument type
atomic	*jobque::Atomic64* ? const implicit
value	int64 const

Set atomic integer value.

get (*atomic: jobque::Atomic64? const implicit*)

get returns int64

argument	argument type
atomic	*jobque::Atomic64* ? const implicit

Get atomic integer value.

inc (*atomic: jobque::Atomic64? const implicit*)

inc returns int64

argument	argument type
atomic	*jobque::Atomic64* ? const implicit

Increase atomic integer value and returns result.

dec (*atomic: jobque::Atomic64? const implicit*)

dec returns int64

argument	argument type
atomic	*jobque::Atomic64* ? const implicit

Decrease atomic integer value and returns result.

CHAPTER

EIGHTEEN

BOOST PACKAGE FOR JOBS AND THREADS

The JOBQUE boost module implements collection of helper macros and functions to accompany *JOBQUE*.

All functions and symbols are in "jobque_boost" module, use require to get access to it.

```
require daslib/jobque_boost
```

18.1 Function annotations

NewJobMacro

this macro handles *new_job* and *new_thread* calls. the call is replaced with *new_job_invoke* and *new_thread_invoke* accordingly. a cloning infastructure is generated for the lambda, which is invoked in the new context.

18.2 Invocations

- *new_job (l:lambda<> -const) : void*
- *new_thread (l:lambda<> -const) : void*

new_job (*l: lambda<>*)

argument	argument type
l	lambda<>

Create a new job.

- new context is cloned from the current context.
- lambda is cloned to the new context.
- new job is added to the job queue.
- once new job is invoked, lambda is invoked on the new context on the job thread.

new_thread (*l: lambda<>*)

argument	argument type
l	lambda<>

Create a new thread

- new context is cloned from the current context.
- lambda is cloned to the new context.
- new thread is created.
- lambda is invoked on the new context on the new thread.

18.3 Iteration

- *for_each (channel:jobque::Channel? const;blk:block<(res:auto(TT) const#):void> const) : auto*
- *each (channel:jobque::Channel? -const;tinfo:auto(TT) const) : auto*

for_each (*channel: jobque::Channel? const; blk: block<(res:auto(TT) const#):void> const*)

for_each returns auto

> **Warning:** This function is deprecated.

argument	argument type
channel	*jobque::Channel* ? const
blk	block<(res:auto(TT) const#):void> const

reads input from the channel (in order it was pushed) and invokes the block on each input. stops once channel is depleted (internal entry counter is 0) this can happen on multiple threads or jobs at the same time.

each (*channel: jobque::Channel?; tinfo: auto(TT) const*)

each returns auto

> **Warning:** This function is deprecated.

argument	argument type
channel	*jobque::Channel* ?
tinfo	auto(TT) const

this iterator is used to iterate over the channel in order it was pushed. iterator stops once channel is depleted (internal entry counter is 0) iteration can happen on multiple threads or jobs at the same time.

18.4 Passing data

- *push_clone (channel:jobque::Channel? const;data:auto(TT) const) : auto*
- *push (channel:jobque::Channel? const;data:auto? const) : auto*

push_clone (*channel: jobque::Channel? const; data: auto(TT) const*)

push_clone returns auto

argument	argument type
channel	*jobque::Channel* ? const
data	auto(TT) const

clones data and pushed value to the channel (at the end)

push (*channel: jobque::Channel? const; data: auto? const*)

push returns auto

argument	argument type
channel	*jobque::Channel* ? const
data	auto? const

pushes value to the channel (at the end)

18.5 Internal capture details

- *capture_jobque_channel (ch:jobque::Channel? const) : jobque::Channel?*
- *capture_jobque_job_status (js:jobque::JobStatus? const) : jobque::JobStatus?*
- *release_capture_jobque_channel (ch:jobque::Channel? const) : void*
- *release_capture_jobque_job_status (js:jobque::JobStatus? const) : void*

capture_jobque_channel (*ch: jobque::Channel? const*)

capture_jobque_channel returns *jobque::Channel* ?

argument	argument type
ch	*jobque::Channel* ? const

this function is used to capture a channel that is used by the jobque.

capture_jobque_job_status (*js: jobque::JobStatus? const*) capture_jobque_job_status returns *jobque::JobStatus* ?

argument	argument type
js	*jobque::JobStatus* ? const

this function is used to capture a job status that is used by the jobque.

release_capture_jobque_channel (*ch: jobque::Channel? const*)

argument	argument type
ch	*jobque::Channel* ? const

this function is used to release a channel that is used by the jobque.

release_capture_jobque_job_status (*js: jobque::JobStatus? const*)

argument	argument type
js	*jobque::JobStatus* ? const

this function is used to release a job status that is used by the jobque.

18.6 Uncategorized

capture_jobque_lock_box (*js: jobque::LockBox? const*)

capture_jobque_lock_box returns *jobque::LockBox* ?

argument	argument type
js	*jobque::LockBox* ? const

this function is used to capture a lock box that is used by the jobque.

release_capture_jobque_lock_box (*js: jobque::LockBox? const*)

argument	argument type
js	*jobque::LockBox* ? const

this function is used to release a lock box that is used by the jobque.

gather (*ch: jobque::Channel? const; blk: block<(arg:auto(TT) const#):void> const*) gather returns auto

argument	argument type
ch	*jobque::Channel* ? const
blk	block<(arg:auto(TT) const#):void> const

reads input from the channel (in order it was pushed) and invokes the block on each input. afterwards input is consumed

gather_ex (*ch: jobque::Channel? const; blk: block<(arg:auto(TT) const#;info:rtti::TypeInfo const? const;var ctx:rtti::Context -const):void> const*)

gather_ex returns auto

argument	argument type
ch	*jobque::Channel* ? const
blk	block<(arg:auto(TT) const#;info: *rtti::TypeInfo* const? const;ctx: *rtti::Context*):void> const

reads input from the channel (in order it was pushed) and invokes the block on each input. afterwards input is consumed

gather_and_forward (*ch: jobque::Channel? const; toCh: jobque::Channel? const; blk: block<(arg:auto(TT) const#):void> const*)

gather_and_forward returns auto

argument	argument type
ch	*jobque::Channel* ? const
toCh	*jobque::Channel* ? const
blk	block<(arg:auto(TT) const#):void> const

reads input from the channel (in order it was pushed) and invokes the block on each input. afterwards input is consumed

peek (*ch: jobque::Channel? const; blk: block<(arg:auto(TT) const#):void> const*)

peek returns auto

argument	argument type
ch	*jobque::Channel* ? const
blk	block<(arg:auto(TT) const#):void> const

reads input from the channel (in order it was pushed) and invokes the block on each input. afterwards input is not consumed

for_each_clone (*channel: jobque::Channel? const; blk: block<(res:auto(TT) const#):void> const*) for_each_clone returns auto

argument	argument type
channel	*jobque::Channel* ? const
blk	block<(res:auto(TT) const#):void> const

reads input from the channel (in order it was pushed) and invokes the block on each input. stops once channel is depleted (internal entry counter is 0) this can happen on multiple threads or jobs at the same time.

pop_one (*channel: jobque::Channel? const; blk: block<(res:auto(TT) const#):void> const*)

pop_one returns auto

> **Warning:** This function is deprecated.

argument	argument type
channel	*jobque::Channel* ? const
blk	block<(res:auto(TT) const#):void> const

reads one command from channel

pop_and_clone_one (*channel: jobque::Channel? const; blk: block<(res:auto(TT) const#):void> const*)

pop_and_clone_one returns auto

argument	argument type
channel	*jobque::Channel* ? const
blk	block<(res:auto(TT) const#):void> const

reads one command from channel

push_batch_clone (*channel: jobque::Channel? const; data: array<auto(TT)> const*)

push_batch_clone returns auto

argument	argument type
channel	*jobque::Channel* ? const
data	array<auto(TT)> const

clones data and pushed values to the channel (at the end)

push_batch (*channel: jobque::Channel? const; data: array<auto?> const*)

push_batch returns auto

argument	argument type
channel	*jobque::Channel* ? const
data	array<auto?> const

pushes values to the channel (at the end)

set (*box: jobque::LockBox? const; data: auto(TT) const*)

set returns auto

argument	argument type
box	*jobque::LockBox* ? const
data	auto(TT) const

sets value to the lock box

set (*box: jobque::LockBox? const; data: auto? const*)

set returns auto

argument	argument type
box	*jobque::LockBox* ? const
data	auto? const

sets value to the lock box

get (*box: jobque::LockBox? const; blk: block<(res:auto(TT) const#):void> const*)

get returns auto

argument	argument type
box	*jobque::LockBox* ? const
blk	block<(res:auto(TT) const#):void> const

reads value from the lock box and invokes the block on it

update (*box: jobque::LockBox? const; blk: block<(var res:auto(TT)# -const):void> const*)

update returns auto

argument	argument type
box	*jobque::LockBox* ? const
blk	block<(res:auto(TT)#):void> const

update value in the lock box and invokes the block on it

clear (*box: jobque::LockBox? const; type_: auto(TT) const*)

clear returns auto

argument	argument type
box	*jobque::LockBox* ? const
type_	auto(TT) const

clear value from the lock box

each_clone (*channel: jobque::Channel?; tinfo: auto(TT) const*)

each_clone returns auto

argument	argument type
channel	*jobque::Channel* ?
tinfo	auto(TT) const

this iterator is used to iterate over the channel in order it was pushed. iterator stops once channel is depleted (internal entry counter is 0) iteration can happen on multiple threads or jobs at the same time.

CHAPTER

NINETEEN

CROSS-CONTEXT EVALUATION HELPERS

The apply_in_context module exposes single [apply_in_context] annotation.

All functions and symbols are in “apply_in_context” module, use require to get access to it.

```
require daslib/apply_in_context
```

19.1 Function annotations

apply_in_context

[apply_in_context] function annotation. Function is modified, so that it is called in the debug agent context, specified in the annotation. If specified context is not insalled, panic is called.

For example:: [apply_in_context(opengl_cache)] def public cache_font(name:string implicit) : Font?

. . .

. . . let font = cache_font(“Arial”) // call invoked in the “opengl_cache” debug agent context

CHAPTER

TWENTY

JSON MANIPULATION LIBRARY

The JSON module implements JSON parser and serialization routines. See *JHSON* <*www.json.org*> for details.

All functions and symbols are in "json" module, use require to get access to it.

```
require daslib/json
```

20.1 Type aliases

JsValue is a variant type

_object	table<string;JsonValue?>
_array	array<JsonValue?>
_string	string
_number	double
_bool	bool
_null	void?

Single JSON element.

Token is a variant type

_string	string
_number	double
_bool	bool
_null	void?
_symbol	int
_error	string

JSON input stream token.

JsonValue

JsonValue fields are

value	*JsValue*

JSON value, wraps any JSON element.

20.2 Value conversion

- *JV (v:string const) : json::JsonValue?*
- *JV (v:double const) : json::JsonValue?*
- *JV (v:bool const) : json::JsonValue?*
- *JVNull () : json::JsonValue?*
- *JV (v:table<string;json::JsonValue?> -const) : json::JsonValue?*
- *JV (v:array<json::JsonValue?> -const) : json::JsonValue?*

JV (*v: string const*)

JV returns *json::JsonValue* ?

argument	argument type
v	string const

Creates *JsonValue* out of value.

JV (*v: double const*)

JV returns *json::JsonValue* ?

argument	argument type
v	double const

Creates *JsonValue* out of value.

JV (*v: bool const*)

JV returns *json::JsonValue* ?

argument	argument type
v	bool const

Creates *JsonValue* out of value.

JVNull ()

JVNull returns *json::JsonValue* ?

Creates *JsonValue* representing *null*.

JV (*v: table<string;json::JsonValue?>*)

JV returns *json::JsonValue* ?

argument	argument type
v	table<string; *json::JsonValue* ?>

Creates *JsonValue* out of value.

JV (*v: array<json::JsonValue?>*)

JV returns *json::JsonValue* ?

argument	argument type
v	array< *json::JsonValue* ?>

Creates *JsonValue* out of value.

20.3 Read and write

- *read_json (text:string const implicit;error:string& -const) : json::JsonValue?*
- *read_json (text:array<uint8> const;error:string& -const) : json::JsonValue?*
- *write_json (val:json::JsonValue? const) : string*
- *write_json (val:json::JsonValue? const#) : string*

read_json (*text: string const implicit; error: string&*)

read_json returns *json::JsonValue* ?

argument	argument type
text	string const implicit
error	string&

reads JSON from the *text* string. if *error* is not empty, it contains the parsing error message.

read_json (*text: array<uint8> const; error: string&*)

read_json returns *json::JsonValue* ?

argument	argument type
text	array<uint8> const
error	string&

reads JSON from the *text* string. if *error* is not empty, it contains the parsing error message.

write_json (*val: json::JsonValue? const*)

write_json returns string

argument	argument type
val	*json::JsonValue* ? const

Overload accepting temporary type

write_json (*val: json::JsonValue? const#*)

write_json returns string

argument	argument type
val	*json::JsonValue* ? const#

Overload accepting temporary type

20.4 JSON properties

- *set_no_trailing_zeros (value:bool const) : bool const*
- *set_no_empty_arrays (value:bool const) : bool const*
- *set_allow_duplicate_keys (value:bool const) : bool const*

set_no_trailing_zeros (*value: bool const*)

set_no_trailing_zeros returns bool const

argument	argument type
value	bool const

if *value* is true, then numbers are written without trailing zeros.

set_no_empty_arrays (*value: bool const*) set_no_empty_arrays returns bool const

argument	argument type
value	bool const

if *value* is true, then empty arrays are not written at all

set_allow_duplicate_keys (*value: bool const*)

set_allow_duplicate_keys returns bool const

argument	argument type
value	bool const

if *value* is true, then duplicate keys are allowed in objects. the later key overwrites the earlier one.

20.5 Broken JSON

- *try_fixing_broken_json (bad:string -const) : string*

try_fixing_broken_json (*bad: string*)

try_fixing_broken_json returns string

argument	argument type
bad	string

fixes broken json. so far supported 1. "string" + "string" string concatination 2. "text "nested text" text" nested quotes 3. extra , at the end of object or array 4. /uXXXXXX sequences in the middle of white space

CHAPTER

TWENTYONE

BOOST PACKAGE FOR JSON

The JSON boost module implements collection of helper macros and functions to accompany *JSON*.

All functions and symbols are in "json_boost" module, use require to get access to it.

```
require daslib/json_boost
```

21.1 Reader macros

json

This macro implements embedding of the JSON object into the program:: var jsv = %json~ {

"name": "main_window", "value": 500, "size": [1,2,3]

} %%

21.2 Variant macros

better_json

This macro is used to implement *is json_value* and *as json_value* runtime checks. It essencially substitutes *value as name* with *value.value as name* and *value is name* with *value.value is name*.

21.3 Value conversion

- *JV (v:float const) : json::JsonValue?*
- *JV (v:int const) : json::JsonValue?*
- *JV (v:bitfield const) : json::JsonValue?*
- *JV (val:int8 const) : json::JsonValue?*
- *JV (val:uint8 const) : json::JsonValue?*
- *JV (val:int16 const) : json::JsonValue?*
- *JV (val:uint16 const) : json::JsonValue?*
- *JV (val:uint const) : json::JsonValue?*
- *JV (val:int64 const) : json::JsonValue?*

- *JV (val:uint64 const) : json::JsonValue?*
- *from_JV (v:json::JsonValue explicit? const;ent:auto(EnumT) const;defV:EnumT const) : EnumT*
- *from_JV (v:json::JsonValue explicit? -const;ent:string const;defV:string const) : auto*
- *from_JV (v:json::JsonValue explicit? -const;ent:bool const;defV:bool const) : auto*
- *from_JV (v:json::JsonValue explicit? -const;ent:float const;defV:float const) : auto*
- *from_JV (v:json::JsonValue explicit? -const;ent:double const;defV:double const) : auto*
- *from_JV (v:json::JsonValue explicit? -const;ent:int const;defV:int const) : auto*
- *from_JV (v:json::JsonValue explicit? -const;ent:uint const;defV:uint const) : auto*
- *from_JV (v:json::JsonValue explicit? -const;ent:int64 const;defV:int64 const) : auto*
- *from_JV (v:json::JsonValue explicit? -const;ent:uint64 const;defV:uint64 const) : auto*
- *from_JV (v:json::JsonValue explicit? -const;ent:int8 const;defV:int8 const) : auto*
- *from_JV (v:json::JsonValue explicit? -const;ent:uint8 const;defV:uint8 const) : auto*
- *from_JV (v:json::JsonValue explicit? -const;ent:int16 const;defV:int16 const) : auto*
- *from_JV (v:json::JsonValue explicit? -const;ent:uint16 const;defV:uint16 const) : auto*
- *from_JV (v:json::JsonValue explicit? -const;ent:bitfield const;defV:bitfield const) : auto*
- *JV (v:auto(VecT) const) : auto*
- *from_JV (v:json::JsonValue explicit? -const;ent:auto(VecT) const;defV:VecT const) : auto*
- *from_JV (v:json::JsonValue explicit? -const;anything:table<auto(KT);auto(VT)> const) : auto*
- *from_JV (v:json::JsonValue explicit? -const;anything:auto(TT) const) : auto*
- *JV (value:auto const) : json::JsonValue?*
- *JV (val1:auto const;val2:auto const) : json::JsonValue?*
- *JV (val1:auto const;val2:auto const;val3:auto const) : json::JsonValue?*
- *JV (val1:auto const;val2:auto const;val3:auto const;val4:auto const) : json::JsonValue?*
- *JV (val1:auto const;val2:auto const;val3:auto const;val4:auto const;val5:auto const) : json::JsonValue?*
- *JV (val1:auto const;val2:auto const;val3:auto const;val4:auto const;val5:auto const;val6:auto const) : json::JsonValue?*
- *JV (val1:auto const;val2:auto const;val3:auto const;val4:auto const;val5:auto const;val6:auto const;val7:auto const) : json::JsonValue?*
- *JV (val1:auto const;val2:auto const;val3:auto const;val4:auto const;val5:auto const;val6:auto const;val7:auto const;val8:auto const) : json::JsonValue?*
- *JV (val1:auto const;val2:auto const;val3:auto const;val4:auto const;val5:auto const;val6:auto const;val7:auto const;val8:auto const;val9:auto const) : json::JsonValue?*
- *JV (val1:auto const;val2:auto const;val3:auto const;val4:auto const;val5:auto const;val6:auto const;val7:auto const;val8:auto const;val9:auto const;val10:auto const) : json::JsonValue?*

JV (*v: float const*)

JV returns *json::JsonValue* ?

argument	argument type
v	float const

Creates *JsonValue* out of value.

JV (*v: int const*)

JV returns *json::JsonValue* ?

argument	argument type
v	int const

Creates *JsonValue* out of value.

JV (*v: bitfield const*)

JV returns *json::JsonValue* ?

argument	argument type
v	bitfield<> const

Creates *JsonValue* out of value.

JV (*val: int8 const*)

JV returns *json::JsonValue* ?

argument	argument type
val	int8 const

Creates *JsonValue* out of value.

JV (*val: uint8 const*)

JV returns *json::JsonValue* ?

argument	argument type
val	uint8 const

Creates *JsonValue* out of value.

JV (*val: int16 const*)

JV returns *json::JsonValue* ?

argument	argument type
val	int16 const

Creates *JsonValue* out of value.

JV (*val: uint16 const*)

JV returns *json::JsonValue* ?

argument	argument type
val	uint16 const

Creates *JsonValue* out of value.

JV (*val: uint const*)

JV returns *json::JsonValue* ?

argument	argument type
val	uint const

Creates *JsonValue* out of value.

JV (*val: int64 const*)

JV returns *json::JsonValue* ?

argument	argument type
val	int64 const

Creates *JsonValue* out of value.

JV (*val: uint64 const*)

JV returns *json::JsonValue* ?

argument	argument type
val	uint64 const

Creates *JsonValue* out of value.

from_JV (*v: json::JsonValue explicit? const; ent: auto(EnumT) const; defV: EnumT const*)

from_JV returns EnumT

argument	argument type
v	*json::JsonValue* ? const
ent	auto(EnumT) const
defV	EnumT const

Parse a JSON value and return the corresponding native value.

from_JV (*v: json::JsonValue explicit?; ent: string const; defV: string const*)

from_JV returns auto

argument	argument type
v	*json::JsonValue* ?
ent	string const
defV	string const

Parse a JSON value and return the corresponding native value.

from_JV (*v: json::JsonValue explicit?; ent: bool const; defV: bool const*)

from_JV returns auto

argument	argument type
v	*json::JsonValue* ?
ent	bool const
defV	bool const

Parse a JSON value and return the corresponding native value.

from_JV (*v: json::JsonValue explicit?; ent: float const; defV: float const*)

from_JV returns auto

argument	argument type
v	*json::JsonValue* ?
ent	float const
defV	float const

Parse a JSON value and return the corresponding native value.

from_JV (*v: json::JsonValue explicit?; ent: double const; defV: double const*)

from_JV returns auto

argument	argument type
v	*json::JsonValue* ?
ent	double const
defV	double const

Parse a JSON value and return the corresponding native value.

from_JV (*v: json::JsonValue explicit?; ent: int const; defV: int const*)

from_JV returns auto

argument	argument type
v	*json::JsonValue* ?
ent	int const
defV	int const

Parse a JSON value and return the corresponding native value.

from_JV (*v: json::JsonValue explicit?; ent: uint const; defV: uint const*)

from_JV returns auto

argument	argument type
v	*json::JsonValue* ?
ent	uint const
defV	uint const

Parse a JSON value and return the corresponding native value.

from_JV (*v: json::JsonValue explicit?; ent: int64 const; defV: int64 const*)

from_JV returns auto

argument	argument type
v	*json::JsonValue* ?
ent	int64 const
defV	int64 const

Parse a JSON value and return the corresponding native value.

from_JV (*v: json::JsonValue explicit?; ent: uint64 const; defV: uint64 const*)

from_JV returns auto

argument	argument type
v	*json::JsonValue* ?
ent	uint64 const
defV	uint64 const

Parse a JSON value and return the corresponding native value.

from_JV (*v: json::JsonValue explicit?; ent: int8 const; defV: int8 const*)

from_JV returns auto

argument	argument type
v	*json::JsonValue* ?
ent	int8 const
defV	int8 const

Parse a JSON value and return the corresponding native value.

from_JV (*v: json::JsonValue explicit?; ent: uint8 const; defV: uint8 const*)

from_JV returns auto

argument	argument type
v	*json::JsonValue* ?
ent	uint8 const
defV	uint8 const

Parse a JSON value and return the corresponding native value.

from_JV (*v: json::JsonValue explicit?; ent: int16 const; defV: int16 const*)

from_JV returns auto

argument	argument type
v	*json::JsonValue* ?
ent	int16 const
defV	int16 const

Parse a JSON value and return the corresponding native value.

from_JV (*v: json::JsonValue explicit?; ent: uint16 const; defV: uint16 const*)

from_JV returns auto

argument	argument type
v	*json::JsonValue* ?
ent	uint16 const
defV	uint16 const

Parse a JSON value and return the corresponding native value.

from_JV (*v: json::JsonValue explicit?; ent: bitfield const; defV: bitfield const*)

from_JV returns auto

argument	argument type
v	*json::JsonValue* ?
ent	bitfield<> const
defV	bitfield<> const

Parse a JSON value and return the corresponding native value.

JV (*v: auto(VecT) const*)

JV returns auto

argument	argument type
v	auto(VecT) const

Creates *JsonValue* out of value.

from_JV (*v: json::JsonValue explicit?; ent: auto(VecT) const; defV: VecT const*)

from_JV returns auto

argument	argument type
v	*json::JsonValue* ?
ent	auto(VecT) const
defV	VecT const

Parse a JSON value and return the corresponding native value.

from_JV (*v: json::JsonValue explicit?; anything: table<auto(KT);auto(VT)> const*)

from_JV returns auto

argument	argument type
v	*json::JsonValue* ?
anything	table<auto(KT);auto(VT)> const

Parse a JSON value and return the corresponding native value.

from_JV (*v: json::JsonValue explicit?; anything: auto(TT) const*)

from_JV returns auto

argument	argument type
v	*json::JsonValue* ?
anything	auto(TT) const

Parse a JSON value and return the corresponding native value.

JV (*value: auto const*)

JV returns *json::JsonValue* ?

argument	argument type
value	auto const

Creates *JsonValue* out of value.

JV (*val1: auto const; val2: auto const*)

JV returns *json::JsonValue* ?

argument	argument type
val1	auto const
val2	auto const

Creates *JsonValue* out of value.

JV (*val1: auto const; val2: auto const; val3: auto const*)

JV returns *json::JsonValue* ?

argument	argument type
val1	auto const
val2	auto const
val3	auto const

Creates *JsonValue* out of value.

JV (*val1: auto const; val2: auto const; val3: auto const; val4: auto const*)

JV returns *json::JsonValue* ?

argument	argument type
val1	auto const
val2	auto const
val3	auto const
val4	auto const

Creates *JsonValue* out of value.

JV (*val1: auto const; val2: auto const; val3: auto const; val4: auto const; val5: auto const*)

JV returns *json::JsonValue* ?

argument	argument type
val1	auto const
val2	auto const
val3	auto const
val4	auto const
val5	auto const

Creates *JsonValue* out of value.

JV (*val1: auto const; val2: auto const; val3: auto const; val4: auto const; val5: auto const; val6: auto const*)

JV returns *json::JsonValue* ?

argument	argument type
val1	auto const
val2	auto const
val3	auto const
val4	auto const
val5	auto const
val6	auto const

Creates *JsonValue* out of value.

JV (*val1: auto const; val2: auto const; val3: auto const; val4: auto const; val5: auto const; val6: auto const; val7: auto const*)

JV returns *json::JsonValue* ?

argument	argument type
val1	auto const
val2	auto const
val3	auto const
val4	auto const
val5	auto const
val6	auto const
val7	auto const

Creates *JsonValue* out of value.

JV (*val1: auto const; val2: auto const; val3: auto const; val4: auto const; val5: auto const; val6: auto const; val7: auto const; val8: auto const*)

JV returns *json::JsonValue* ?

argument	argument type
val1	auto const
val2	auto const
val3	auto const
val4	auto const
val5	auto const
val6	auto const
val7	auto const
val8	auto const

Creates *JsonValue* out of value.

JV (*val1: auto const; val2: auto const; val3: auto const; val4: auto const; val5: auto const; val6: auto const; val7: auto const; val8: auto const; val9: auto const*)

JV returns *json::JsonValue* ?

argument	argument type
val1	auto const
val2	auto const
val3	auto const
val4	auto const
val5	auto const
val6	auto const
val7	auto const
val8	auto const
val9	auto const

Creates *JsonValue* out of value.

JV (*val1: auto const; val2: auto const; val3: auto const; val4: auto const; val5: auto const; val6: auto const; val7: auto const; val8: auto const; val9: auto const; val10: auto const*)

JV returns *json::JsonValue* ?

argument	argument type
val1	auto const
val2	auto const
val3	auto const
val4	auto const
val5	auto const
val6	auto const
val7	auto const
val8	auto const
val9	auto const
val10	auto const

Creates *JsonValue* out of value.

CHAPTER

TWENTYTWO

REGULAR EXPRESSION LIBRARY

The *experimental* REGEX module implement regular expression parser and pattern matching functionality.

Currently its in very early stage and implements only very few basic regex operations.

All functions and symbols are in "regex" module, use require to get access to it.

```
require daslib/regex
```

22.1 Type aliases

CharSet = uint[8]

Bit array which represents an 8-bit character set.

ReGenRandom = iterator<uint>

random generator input for the regular expression generation

MaybeReNode is a variant type

value	*regex::ReNode* ?
nothing	void?

Single regular expression node or nothing.

22.2 Enumerations

ReOp

Char	0
Set	1
Any	2
Eos	3
Group	4
Plus	5
Star	6
Question	7
Concat	8
Union	9

Type of regular expression operation.

ReNode

ReNode fields are

op	*regex::ReOp*
id	int
fun2	function<(regex: *regex::Regex* ;node: *regex::ReNode* ?;str:uint8? const):uint8?>
gen2	function<(node: *regex::ReNode* ?;rnd: *ReGenRandom* ;str: *strings::StringBuilderWriter*):void>
at	range
text	string
textLen	int
all	array< *regex::ReNode* ?>
left	*regex::ReNode* ?
right	*regex::ReNode* ?
subexpr	*regex::ReNode* ?
next	*regex::ReNode* ?
cset	*CharSet*
index	int
tail	uint8?

Single node in regular expression parsing tree.

Regex

Regex fields are

root	*regex::ReNode* ?
match	uint8?
groups	array<tuple<range;string>>
earlyOut	*CharSet*
canEarlyOut	bool

Regular expression.

22.3 Compilation and validation

- *visit_top_down (node:regex::ReNode? -const;blk:block<(var n:regex::ReNode? -const):void> const) : void*
- *is_valid (re:regex::Regex -const) : bool*
- *regex_compile (re:regex::Regex -const;expr:string const) : bool*
- *regex_compile (expr:string const) : regex::Regex*
- *regex_compile (re:regex::Regex -const) : regex::Regex*
- *regex_debug (regex:regex::Regex const) : void*
- *debug_set (cset:uint const[8]) : void*

visit_top_down (*node: regex::ReNode?; blk: block<(var n:regex::ReNode? -const):void> const*)

argument	argument type
node	*regex::ReNode* ?
blk	block<(n: *regex::ReNode* ?):void> const

visits parsed regular expression tree, parents first

is_valid (*re: Regex*)

is_valid returns bool

argument	argument type
re	*regex::Regex*

returns *true* if enumeration compiled correctly

regex_compile (*re: Regex; expr: string const*)

regex_compile returns bool

argument	argument type
re	*regex::Regex*
expr	string const

Compile regular expression. Validity of the compiled expression is checked by *is_valid*.

regex_compile (*expr: string const*)

regex_compile returns *regex::Regex*

argument	argument type
expr	string const

Compile regular expression. Validity of the compiled expression is checked by *is_valid*.

regex_compile (*re: Regex*)

regex_compile returns *regex::Regex*

argument	argument type
re	*regex::Regex*

Compile regular expression. Validity of the compiled expression is checked by *is_valid*.

regex_debug (*regex: Regex const*)

argument	argument type
regex	*regex::Regex* const

Prints regular expression and its related information in human readable form.

debug_set (*cset: CharSet*)

argument	argument type
cset	*CharSet*

Prints character set in human readable form.

22.4 Access

- *regex_group (regex:regex::Regex const;index:int const;match:string const) : string*
- *regex_foreach (regex:regex::Regex -const;str:string const;blk:block<(at:range const):bool> const) : void*

regex_group (*regex: Regex const; index: int const; match: string const*)

regex_group returns string

argument	argument type
regex	*regex::Regex* const
index	int const
match	string const

Returns string for the given group index and match result.

regex_foreach (*regex: Regex; str: string const; blk: block<(at:range const):bool> const*)

argument	argument type
regex	*regex::Regex*
str	string const
blk	block<(at:range const):bool> const

Iterates through all matches for the given regular expression in *str*.

22.5 Match

- *regex_match (regex:regex::Regex -const;str:string const;offset:int const) : int*

regex_match (*regex: Regex; str: string const; offset: int const*)

regex_match returns int

argument	argument type
regex	*regex::Regex*
str	string const
offset	int const

Returns first match for the regular expression in *str*. If *offset* is specified, first that many number of symbols will not be matched.

22.6 Generation

- *re_gen_get_rep_limit () : uint*
- *re_gen (re:regex::Regex -const;rnd:iterator<uint> -const) : string*

re_gen_get_rep_limit ()

re_gen_get_rep_limit returns uint

repetition limit for the '+' and '*' operations of the regex generation

re_gen (*re: Regex; rnd: ReGenRandom*)

re_gen returns string

argument	argument type
re	*regex::Regex*
rnd	*ReGenRandom*

generates random string which would match regular expression

22.7 Uncategorized

regex_replace (*regex: Regex; str: string const; blk: block<(at:string const):string> const*)

regex_replace returns string const

argument	argument type
regex	*regex::Regex*
str	string const
blk	block<(at:string const):string> const

Iterates through all matches for the given regular expression in *str*.

CHAPTER

TWENTYTHREE

BOOST PACKAGE FOR REGEX

The REGEX boost module implements collection of helper macros and functions to accompany *REGEX*.

All functions and symbols are in "regex_boost" module, use require to get access to it.

```
require daslib/regex_boost
```

23.1 Reader macros

regex

This macro implements embedding of the REGEX object into the AST:: var op_regex <- %regex~operator[^a-**zA-Z_**]%%

Regex is compiled at the time of parsing, and the resulting object is embedded into the AST.

CHAPTER

TWENTYFOUR

DOCUMENTATION GENERATOR

The RST module exposes collection of helper routines to automatically generate Daslang reStructuredText documentation.

All functions and symbols are in “rst” module, use require to get access to it.

```
require daslib/rst
```

DocGroup

DocGroup fields are

name	string
func	array< *ast::Function* ?>
hidden	bool

Group of functions with shared category.

24.1 Document writers

- *document (name:string const;mod:rtti::Module? const;fname:string const;substname:string const;groups:array<rst::DocGroup> const) : void*

document (*name: string const; mod: rtti::Module? const; fname: string const; substname: string const; groups: array<rst::DocGroup> const*)

argument	argument type
name	string const
mod	*rtti::Module* ? const
fname	string const
substname	string const
groups	array< *rst::DocGroup* > const

Document single module given list of *DocGropus*. This will generate RST file with documentation for the module. Functions which do not match any *DocGroup* will be placed in the *Uncategorized* group.

24.2 Group operations

- *group_by_regex (name:string const;mod:rtti::Module? const;reg:regex::Regex -const) : rst::DocGroup*
- *hide_group (group:rst::DocGroup -const) : rst::DocGroup*

group_by_regex(*name: string const; mod: rtti::Module? const; reg: Regex*)

group_by_regex returns *rst::DocGroup*

argument	argument type
name	string const
mod	*rtti::Module* ? const
reg	*regex::Regex*

Creates a group of functions with shared category. Functions will be added to the group if they match the regular expression.

hide_group(*group: DocGroup*)

hide_group returns *rst::DocGroup*

argument	argument type
group	*rst::DocGroup*

Marks the group as hidden.

CHAPTER

TWENTYFIVE

APPLY REFLECTION PATTERN

Apply module implements *apply* pattern, i.e. static reflection dispatch for structures and other data types.

All functions and symbols are in "apply" module, use require to get access to it.

```
require daslib/apply
```

25.1 Call macros

apply

This macro implements the apply() pattern. The idea is that for each entry in the structure, variant, or tuple, the block will be invoked. Both element name, and element value are passed to the block. For example

> **struct Bar** x, y : float
>
> **apply([[Bar x=1.,y=2.]]) <| $ (name:string; field)** print("{name} = {field} ")

Would print x = 1.0 y = 2.0

CHAPTER

TWENTYSIX

MISCELANIOUS ALGORITHMS

The ALGORITHM module exposes collection of miscellaneous array manipulation algorithms.

All functions and symbols are in “algorithm” module, use require to get access to it.

```
require daslib/algorithm
```

26.1 Search

- *lower_bound (a:array<auto(TT)> const;f:int const;l:int const;val:TT const -&) : auto*
- *lower_bound (a:array<auto(TT)> const;val:TT const -&) : auto*
- *lower_bound (a:array<auto(TT)> const;f:int const;l:int const;value:TT const -&;less:block<(a:TT const -&;b:TT const -&):bool> const) : auto*
- *lower_bound (a:array<auto(TT)> const;value:TT const -&;less:block<(a:TT const -&;b:TT const -&):bool> const) : auto*
- *binary_search (a:array<auto(TT)> const;val:TT const -&) : auto*
- *binary_search (a:array<auto(TT)> const;f:int const;last:int const;val:TT const -&) : auto*
- *binary_search (a:array<auto(TT)> const;val:TT const -&;less:block<(a:TT const -&;b:TT const -&):bool> const) : auto*
- *binary_search (a:array<auto(TT)> const;f:int const;last:int const;val:TT const -&;less:block<(a:TT const -&;b:TT const -&):bool> const) : auto*
- *lower_bound (a:auto const;f:int const;l:int const;val:auto const) : auto*
- *lower_bound (a:auto const;val:auto const) : auto*
- *lower_bound (a:auto const;f:int const;l:int const;val:auto(TT) const;less:block<(a:TT const -&;b:TT const -&):bool> const) : auto*
- *lower_bound (a:auto const;val:auto(TT) const;less:block<(a:TT const -&;b:TT const -&):bool> const) : auto*
- *binary_search (a:auto const;val:auto const) : auto*
- *binary_search (a:auto const;f:int const;last:int const;val:auto const) : auto*
- *binary_search (a:auto const;val:auto(TT) const;less:block<(a:TT const -&;b:TT const -&):bool> const) : auto*
- *binary_search (a:auto const;f:int const;last:int const;val:auto(TT) const;less:block<(a:TT const -&;b:TT const -&):bool> const) : auto*

lower_bound(*a: array<auto(TT)> const; f: int const; l: int const; val: TT const*) lower_bound returns auto

argument	argument type
a	array<auto(TT)> const
f	int const
l	int const
val	TT const

Returns an iterator pointing to the first element in the range [first, last) that is not less than (i.e. greater or equal to) value, or last if no such element is found.

lower_bound (*a: array<auto(TT)> const; val: TT const*)

lower_bound returns auto

argument	argument type
a	array<auto(TT)> const
val	TT const

Returns an iterator pointing to the first element in the range [first, last) that is not less than (i.e. greater or equal to) value, or last if no such element is found.

lower_bound (*a: array<auto(TT)> const; f: int const; l: int const; value: TT const; less: block<(a:TT const -&;b:TT const -&):bool> const*)

lower_bound returns auto

argument	argument type
a	array<auto(TT)> const
f	int const
l	int const
value	TT const
less	block<(a:TT const;b:TT const):bool> const

Returns an iterator pointing to the first element in the range [first, last) that is not less than (i.e. greater or equal to) value, or last if no such element is found.

lower_bound (*a: array<auto(TT)> const; value: TT const; less: block<(a:TT const -&;b:TT const -&):bool> const*)

lower_bound returns auto

argument	argument type
a	array<auto(TT)> const
value	TT const
less	block<(a:TT const;b:TT const):bool> const

Returns an iterator pointing to the first element in the range [first, last) that is not less than (i.e. greater or equal to) value, or last if no such element is found.

binary_search (*a: array<auto(TT)> const; val: TT const*)

binary_search returns auto

argument	argument type
a	array<auto(TT)> const
val	TT const

Returns true if an val appears within the range [f, last). Array a must be sorted.

binary_search (*a: array<auto(TT)> const; f: int const; last: int const; val: TT const*)

binary_search returns auto

argument	argument type
a	array<auto(TT)> const
f	int const
last	int const
val	TT const

Returns true if an val appears within the range [f, last). Array a must be sorted.

binary_search (*a: array<auto(TT)> const; val: TT const; less: block<(a:TT const -&;b:TT const -&):bool> const*)

binary_search returns auto

argument	argument type
a	array<auto(TT)> const
val	TT const
less	block<(a:TT const;b:TT const):bool> const

Returns true if an val appears within the range [f, last). Array a must be sorted.

binary_search (*a: array<auto(TT)> const; f: int const; last: int const; val: TT const; less: block<(a:TT const -&;b:TT const -&):bool> const*)

binary_search returns auto

argument	argument type
a	array<auto(TT)> const
f	int const
last	int const
val	TT const
less	block<(a:TT const;b:TT const):bool> const

Returns true if an val appears within the range [f, last). Array a must be sorted.

lower_bound (*a: auto const; f: int const; l: int const; val: auto const*)

lower_bound returns auto

argument	argument type
a	auto const
f	int const
l	int const
val	auto const

Returns an iterator pointing to the first element in the range [first, last) that is not less than (i.e. greater or equal to) value, or last if no such element is found.

lower_bound (*a: auto const; val: auto const*)

lower_bound returns auto

argument	argument type
a	auto const
val	auto const

Returns an iterator pointing to the first element in the range [first, last) that is not less than (i.e. greater or equal to) value, or last if no such element is found.

lower_bound (*a: auto const; f: int const; l: int const; val: auto(TT) const; less: block<(a:TT const -&;b:TT const -&):bool> const*) lower_bound returns auto

argument	argument type
a	auto const
f	int const
l	int const
val	auto(TT) const
less	block<(a:TT const;b:TT const):bool> const

Returns an iterator pointing to the first element in the range [first, last) that is not less than (i.e. greater or equal to) value, or last if no such element is found.

`lower_bound` (*a: auto const; val: auto(TT) const; less: block<(a:TT const -&;b:TT const -&):bool> const*)

lower_bound returns auto

argument	argument type
a	auto const
val	auto(TT) const
less	block<(a:TT const;b:TT const):bool> const

Returns an iterator pointing to the first element in the range [first, last) that is not less than (i.e. greater or equal to) value, or last if no such element is found.

`binary_search` (*a: auto const; val: auto const*)

binary_search returns auto

argument	argument type
a	auto const
val	auto const

Returns true if an val appears within the range [f, last). Array a must be sorted.

`binary_search` (*a: auto const; f: int const; last: int const; val: auto const*)

binary_search returns auto

argument	argument type
a	auto const
f	int const
last	int const
val	auto const

Returns true if an val appears within the range [f, last). Array a must be sorted.

binary_search (*a: auto const; val: auto(TT) const; less: block<(a:TT const -&;b:TT const -&):bool> const*)

binary_search returns auto

argument	argument type
a	auto const
val	auto(TT) const
less	block<(a:TT const;b:TT const):bool> const

Returns true if an val appears within the range [f, last). Array a must be sorted.

binary_search (*a: auto const; f: int const; last: int const; val: auto(TT) const; less: block<(a:TT const -&;b:TT const -&):bool> const*)

binary_search returns auto

argument	argument type
a	auto const
f	int const
last	int const
val	auto(TT) const
less	block<(a:TT const;b:TT const):bool> const

Returns true if an val appears within the range [f, last). Array a must be sorted.

26.2 Array manipulation

- *unique (a:array<auto(TT)> -const) : auto*
- *sort_unique (a:array<auto(TT)> -const) : auto*
- *reverse (a:array<auto> -const) : auto*
- *combine (a:array<auto(TT)> const;b:array<auto(TT)> const) : auto*
- *reverse (a:auto -const) : auto*
- *combine (a:auto const;b:auto const) : auto*

unique (*a: array<auto(TT)>*)

unique returns auto

argument	argument type
a	array<auto(TT)>

Returns array of the elements of a with duplicates removed.

sort_unique (*a: array<auto(TT)>*)

sort_unique returns auto

argument	argument type
a	array<auto(TT)>

Returns array of the elements of a, sorted and with duplicates removed. The elements of a are sorted in ascending order. The resulted array has only unqiue elements.

reverse (*a: array<auto>*)

reverse returns auto

argument	argument type
a	array<auto>

Returns array of the elements of a in reverse order.

combine (*a: array<auto(TT)> const; b: array<auto(TT)> const*)

combine returns auto

argument	argument type
a	array<auto(TT)> const
b	array<auto(TT)> const

Returns array of the elements of a and then b.

reverse (*a: auto*)

reverse returns auto

argument	argument type
a	auto

Returns array of the elements of a in reverse order.

combine (*a: auto const; b: auto const*)

combine returns auto

argument	argument type
a	auto const
b	auto const

Returns array of the elements of a and then b.

26.3 Uncategorized

erase_all (*arr: auto; value: auto const*)

erase_all returns auto

argument	argument type
arr	auto
value	auto const

Erase all elements equal to value from arr

topological_sort (*nodes: array<auto(Node)> const*)

topological_sort returns auto

argument	argument type
nodes	array<auto(Node)> const

Topological sort of a graph. Each node has an id, and set (table with no values) of dependencies. Dependency *before* represents a link from a node, which should appear in the sorted list before the node. Returns a sorted list of nodes.

CHAPTER

TWENTYSEVEN

MISCELANIOUS CONTRACT ANNOTATIONS

The CONTRACTS module exposes collection of type matching contracts.

All functions and symbols are in “contracts” module, use require to get access to it.

```
require daslib/contracts
```

27.1 Function annotations

expect_any_array

[expect_any_array(argname)] contract, which only accepts array<T>, T[], or das`vector<T>

expect_dim

[expect_dim(argname)] contract, which only accepts T[]

expect_any_enum

[expect_any_enum(argname)] contract, which only accepts enumerations

expect_any_bitfield

[expect_any_bitfield(argname)] contract, which only accepts bitfields

expect_any_vector_type

[expect_any_vector_type(argname)] contract, which only accepts vector types, i.e. int2, float3, range, etc

expect_any_struct

[expect_any_struct(argname)] contract, which only accepts structs (byt not classes)

expect_any_numeric

[expect_any_numeric(argname)] contract, which only accepts numeric types (int, float, etc)

expect_any_workhorse

[expect_any_workhorse(argname)] contract, which only accepts workhorse types (int, float, etc) Workhorse types are: bool,int*,uint*,float*,double,range and urange, range64 and urange64, string,enumeration,and non-smart pointers

expect_any_workhorse_raw

[expect_any_workhorse_raw(argname)] contract, which only accepts workhorse types which are raw (not pointer or bool)

expect_any_tuple

[expect_any_tuple(argname)] contract, which only accepts tuples

expect_any_variant

[expected_any_variant(argname)] contract, which only accepts variants

expect_any_function

[expect_any_function(argname)] contract, which only accepts functions

expect_any_lambda

[expect_any_lambda(argname)] contract, which only accepts lambdas

expect_ref

[expect_ref(argname)] contract, which only accepts references

expect_pointer

[expect_pointer(argname)] contract, which only accepts pointers

expect_class

[expect_class(argname)] contract, which only accepts class instances

expect_value_handle

[expect_value_handle(argname)] contract, which only accepts value handles

27.2 Type queries

- *isYetAnotherVectorTemplate (td:smart_ptr<ast::TypeDecl> const) : bool*

isYetAnotherVectorTemplate(*td: TypeDeclPtr*)

isYetAnotherVectorTemplate returns bool

argument	argument type
td	*TypeDeclPtr*

returns true if the given type declaration is a das::vector template bound on C++ side

CHAPTER

TWENTYEIGHT

DEFER AND DEFER_DELETE MACROS

Apply module implements *defer* and *defer_delete* pattern, i.e. ability to attach a bit of code or a delete operation to a finally section of the block, without leaving the context of the code.

All functions and symbols are in "defer" module, use require to get access to it.

```
require daslib/defer
```

28.1 Function annotations

DeferMacro

This macro covnerts defer() <| block expression into {}, and move block to the finally section of the current block

28.2 Call macros

defer_delete

This macro converts defer_delete() expression into {}, and add delete expression to the finally section of the current block

28.3 Defer

- *defer (blk:block<void> const) : void*

defer (*blk: block<void> const*)

argument	argument type
blk	block<> const

defer a block of code. For example:

```
var a = fopen("filename.txt","r")
defer <|
    fclose(a)
```

Will close the file when 'a' is out of scope.

28.4 Stub

- *nada () : void*

nada()

helper function which does nothing and will be optimized out

CHAPTER

TWENTYNINE

IF_NOT_NULL MACRO

The if_not_null module exposes single *if_not_null* pattern.

All functions and symbols are in "if_not_null" module, use require to get access to it.

```
require daslib/if_not_null
```

29.1 Call macros

if_not_null

This macro transforms:

```
ptr |> if_not_null <| call(...)
```

to:

```
var _ptr_var = ptr
if _ptr_var
    call(*_ptr_var,...)
```

CHAPTER

THIRTY

INSTANCE_FUNCTION FUNCTION ANNOTATION

The instance_function module exposes a way to declaratively instance a generic function with particular set of types.

All functions and symbols are in "instance_function" module, use require to get access to it.

```
require daslib/instance_function
```

30.1 Function annotations

instance_function

[instance_function(generic_name,type1=type1r,type2=type2r,...)] macro creates instance of the generic function with a particular set of types. In the followin example body of the function inst will be replaced with body of the function print_zero with type int:

```
def print_zero ( a : auto(TT) )
    print("{[[TT]]}\n")
[export, instance_function(print_zero,TT="int")]
def inst {}
```

CHAPTER

THIRTYONE

DECLTYPE MACRO AND TEMPLATE FUNCTION ANNOTATION

The templates exposes collection of template-like routines for Daslang.

All functions and symbols are in "templates" module, use require to get access to it.

```
require daslib/templates
```

31.1 Function annotations

template

This macro is used to remove unused (template) arguments from the instantiation of the generic function. When [template(x)] is specified, the argument x is removed from the function call, but the type of the instance remains. The call where the function is instanciated is adjusted as well. For example:

```
[template (a), sideeffects]
def boo ( x : int; a : auto(TT) )   // when boo(1,type<int>)
    return "{x}_{typeinfo(typename type<TT>)}"
...
boo(1,type<int>) // will be replaced with boo(1). instace will print "1_int"
```

31.2 Call macros

decltype

This macro returns ast::TypeDecl for the corresponding expression. For example:

```
let x = 1
let y <- decltype(x) // [[TypeDecl() baseType==Type tInt, flags=TypeDeclFlags␣
↪constant | TypeDeclFlags ref]]
```

decltype_noref

This macro returns TypeDecl for the corresponding expression, minus the ref (&) portion.

CHAPTER

THIRTYTWO

TEMPLATE APPLICATION HELPERS

The templates boost module implements collection of helper macros and functions to accompany *AST*.

All functions and symbols are in "templates_boost" module, use require to get access to it.

```
require daslib/templates_boost
```

Template

Template fields are

kaboomVar	table<string;tuple<prefix:string;suffix:string>>
call2name	table<string;string>
field2name	table<string;string>
var2name	table<string;string>
var2expr	table<string;smart_ptr< *ast::Expression* >>
var2exprList	table<string;array<smart_ptr< *ast::Expression* >>>
type2type	table<string;string>
type2etype	table<string; *TypeDeclPtr* >
blockArgName	table<string;string>
annArg	table<string;lambda<(ann: *rtti::AnnotationDeclaration*):void>>
blkArg	table<string;array< *VariablePtr* >>
tag2expr	table<string;smart_ptr< *ast::Expression* >>

This structure contains collection of subsitution rules for a template.

32.1 Call macros

qmacro_expr

This macro implements *qmacro_expr* expression reification. Expected input is a block expression (ExprMakeBlock over ExprBlock). It applies reification rules to the expression, and returns first expression in the block. .. _call-macro-templates_boost-qmacro_variable:

qmacro_variable

This macro implements *qmacro_variable* expression reification. Expected input is are variable name and type expression (type<…>). Result is a new VariablePtr with the matching name and type. .. _call-macro-templates_boost-qmacro_block_to_array:

qmacro_block_to_array

This macro implements *qmacro_block_to_array* expression reification. Expected input is a block expression (ExprMakeBlock over ExprBlock). It applies reification rules to the expression, and returns array with contents of the 'list' section of the block.

qmacro_function

This macro implements *qmacro_function* expression reification. Expected input is a block expression (ExprMakeBlock over ExprBlock). It applies reification rules to the expression, and returns a FunctionPtr. New function matches block signature, as well as the block body. .. _call-macro-templates_boost-qmacro:

qmacro

This macro implements *qmacro* expression reification. It applies reification rules to the expression, and returns direct result of the substitution.

qmacro_method

This macro implements expression reification for class methods.

qmacro_block

This macro implements *qmacro_block* expression reification. Expected input is a block expression (ExprMakeBlock over ExprBlock). It applies reification rules to the expression, and returns unquoted *ExprBlock*. .. _call-macro-templates_boost-qmacro_type:

qmacro_type

This macro implements *qmacro_type* expression reification. Expected input is a type expression (type<…>). Result is TypeDeclPtr of a new type matching subtype of the type expression.

32.2 Template rules

- *kaboomVarField (self:templates_boost::Template -const;name:string const;prefix:string const;suffix:string const) : void*
- *replaceVariable (self:templates_boost::Template -const;name:string const;expr:smart_ptr<ast::Expression> -const) : void*
- *renameVariable (self:templates_boost::Template -const;name:string const;newName:string const) : void*
- *renameVariable (self:templates_boost::Template -const;name:string const;newName:$::das_string const) : void*
- *replaceType (self:templates_boost::Template -const;name:string const;newName:string const) : void*

- *replaceAnnotationArgument (self:templates_boost::Template -const;name:string const;cb:lambda<(var ann:rtti::AnnotationDeclaration -const):void> -const) : void*
- *replaceBlockArgument (self:templates_boost::Template -const;name:string const;newName:string const) : void*

kaboomVarField (*self: Template; name: string const; prefix: string const; suffix: string const*)

argument	argument type
self	*templates_boost::Template*
name	string const
prefix	string const
suffix	string const

Adds a rule to to the template to replace a variable field access with a prefix and suffix. I.e. foo.bar into prefix + bar + suffix

replaceVariable (*self: Template; name: string const; expr: smart_ptr<ast::Expression>*)

argument	argument type
self	*templates_boost::Template*
name	string const
expr	smart_ptr< *ast::Expression* >

Adds a rule to the template to replace a variable with an expression.

renameVariable (*self: Template; name: string const; newName: string const*)

argument	argument type
self	*templates_boost::Template*
name	string const
newName	string const

Adds a rule to the template to rename a variable.

renameVariable (*self: Template; name: string const; newName: das_string const*)

argument	argument type
self	*templates_boost::Template*
name	string const
newName	*builtin::das_string* const

Adds a rule to the template to rename a variable.

replaceType (*self: Template; name: string const; newName: string const*)

argument	argument type
self	*templates_boost::Template*
name	string const
newName	string const

Adds a rule to the template to replace a type alias with another type alias, specified by name.

replaceAnnotationArgument (*self: Template; name: string const; cb: lambda<(var ann:rtti::AnnotationDeclaration -const):void>*)

argument	argument type
self	*templates_boost::Template*
name	string const
cb	lambda<(ann: *rtti::AnnotationDeclaration*):void>

Adds a rule to the template to replace an annotation argument with the result of a callback.

replaceBlockArgument (*self: Template; name: string const; newName: string const*)

argument	argument type
self	*templates_boost::Template*
name	string const
newName	string const

Adds a rule to the template to rename a block argument.

32.3 Template application

- *apply_template (rules:templates_boost::Template -const;at:rtti::LineInfo const;expr:smart_ptr<ast::Expression> -const;forceAt:bool const) : smart_ptr<ast::Expression>*
- *apply_template (at:rtti::LineInfo const;expr:smart_ptr<ast::Expression>& -const;blk:block<(var rules:templates_boost::Template -const):void> const) : smart_ptr<ast::Expression>*
- *apply_template (expr:smart_ptr<ast::Expression>& -const;blk:block<(var rules:templates_boost::Template -const):void> const) : smart_ptr<ast::Expression>*

apply_template (*rules: Template; at: LineInfo const; expr: smart_ptr<ast::Expression>; forceAt: bool const*)

apply_template returns *ExpressionPtr*

argument	argument type
rules	*templates_boost::Template*
at	*rtti::LineInfo* const
expr	smart_ptr< *ast::Expression* >
forceAt	bool const

Applies the template to the given expression. If *forceAt* is set, the resulting expression will have the same line info as 'at'.

apply_template (*at: LineInfo const; expr: smart_ptr<ast::Expression>&; blk: block<(var rules:templates_boost::Template -const):void> const*)

apply_template returns *ExpressionPtr*

argument	argument type
at	*rtti::LineInfo* const
expr	smart_ptr< *ast::Expression* >&
blk	block<(rules: *templates_boost::Template*):void> const

Applies the template to the given expression. If *forceAt* is set, the resulting expression will have the same line info as 'at'.

apply_template (*expr: smart_ptr<ast::Expression>&; blk: block<(var rules:templates_boost::Template -const):void> const*)

apply_template returns *ExpressionPtr*

argument	argument type
expr	smart_ptr< *ast::Expression* >&
blk	block<(rules: *templates_boost::Template*):void> const

Applies the template to the given expression. If *forceAt* is set, the resulting expression will have the same line info as 'at'.

32.4 Expression helpers

- *remove_deref (varname:string const;expr:smart_ptr<ast::Expression> -const) : void*

remove_deref (*varname: string const; expr: smart_ptr<ast::Expression>*)

argument	argument type
varname	string const
expr	smart_ptr< *ast::Expression* >

Removes dereferences of the variable *varname* from the expression. This is typically used when replacing 'workhorse' variable with constant.

32.5 Block helpers

- *unquote_block (expr:smart_ptr<ast::Expression> const) : smart_ptr<ast::ExprBlock>*
- *move_unquote_block (expr:smart_ptr<ast::Expression>& -const) : smart_ptr<ast::ExprBlock>*

unquote_block (*expr: ExpressionPtr*)

unquote_block returns smart_ptr< *ast::ExprBlock* >

argument	argument type
expr	*ExpressionPtr*

Returns the corresponding block subexpression expression form the ExprMakeBlock.

move_unquote_block (*expr: ExpressionPtr*)

move_unquote_block returns smart_ptr< *ast::ExprBlock* >

argument	argument type
expr	*ExpressionPtr*

Moves the corresponding block subexpression expression form the ExprMakeBlock.

32.6 Global variable helpers

- *add_global_var (mod:rtti::Module? const;vname:string const;vat:rtti::LineInfo const;value:smart_ptr<ast::Expression> -const) : bool*
- *add_global_var (mod:rtti::Module? const;vname:string const;typ:smart_ptr<ast::TypeDecl> -const;vat:rtti::LineInfo const;priv:bool const;blk:block<(var v:smart_ptr<ast::Variable> -const):void> const) : bool*
- *add_global_var (mod:rtti::Module? const;vname:string const;typ:smart_ptr<ast::TypeDecl> -const;vat:rtti::LineInfo const;priv:bool const) : bool*
- *add_global_let (mod:rtti::Module? const;vname:string const;vat:rtti::LineInfo const;value:smart_ptr<ast::Expression> -const) : bool*
- *add_global_private_var (mod:rtti::Module? const;vname:string const;vat:rtti::LineInfo const;value:smart_ptr<ast::Expression> -const) : bool*
- *add_global_private_let (mod:rtti::Module? const;vname:string const;vat:rtti::LineInfo const;value:smart_ptr<ast::Expression> -const) : bool*

add_global_var (*mod: rtti::Module? const; vname: string const; vat: LineInfo const; value: ExpressionPtr*)

add_global_var returns bool

argument	argument type
mod	*rtti::Module* ? const
vname	string const
vat	*rtti::LineInfo* const
value	*ExpressionPtr*

Adds global variable to the module, given name and initial value. Global variables type is would be inferred from the initial value. *priv* specifies if the variable is private to the block.

add_global_var (*mod: rtti::Module? const; vname: string const; typ: TypeDeclPtr; vat: LineInfo const; priv: bool const; blk: block<(var v:smart_ptr<ast::Variable> -const):void> const*)

add_global_var returns bool

argument	argument type
mod	*rtti::Module* ? const
vname	string const
typ	*TypeDeclPtr*
vat	*rtti::LineInfo* const
priv	bool const
blk	block<(v: *VariablePtr*):void> const

Adds global variable to the module, given name and initial value. Global variables type is would be inferred from the initial value. *priv* specifies if the variable is private to the block.

add_global_var (*mod: rtti::Module? const; vname: string const; typ: TypeDeclPtr; vat: LineInfo const; priv: bool const*)

add_global_var returns bool

argument	argument type
mod	*rtti::Module* ? const
vname	string const
typ	*TypeDeclPtr*
vat	*rtti::LineInfo* const
priv	bool const

Adds global variable to the module, given name and initial value. Global variables type is would be inferred from the initial value. *priv* specifies if the variable is private to the block.

add_global_let (*mod: rtti::Module? const; vname: string const; vat: LineInfo const; value: ExpressionPtr*)

add_global_let returns bool

argument	argument type
mod	*rtti::Module* ? const
vname	string const
vat	*rtti::LineInfo* const
value	*ExpressionPtr*

Add global variable to the module, given name and initial value. Variable type will be constant.

add_global_private_var (*mod: rtti::Module? const; vname: string const; vat: LineInfo const; value: ExpressionPtr*)

add_global_private_var returns bool

argument	argument type
mod	*rtti::Module* ? const
vname	string const
vat	*rtti::LineInfo* const
value	*ExpressionPtr*

Add global variable to the module, given name and initial value. It will be private.

add_global_private_let (*mod: rtti::Module? const; vname: string const; vat: LineInfo const; value: ExpressionPtr*)

add_global_private_let returns bool

argument	argument type
mod	*rtti::Module* ? const
vname	string const
vat	*rtti::LineInfo* const
value	*ExpressionPtr*

Add global variable to the module, given name and initial value. It will be private, and type will be constant.

32.7 Hygenic names

- *make_unique_private_name (prefix:string const;vat:rtti::LineInfo const) : string*

make_unique_private_name (*prefix: string const; vat: LineInfo const*)

make_unique_private_name returns string

argument	argument type
prefix	string const
vat	*rtti::LineInfo* const

Generates unique private name for the variable, given prefix and line info.

The assumption is that line info is unique for the context of the unique name generation. If it is not, additional measures must be taken to ensure uniqueness of prefix.

32.8 Uncategorized

replaceVarTag (*self: Template; name: string const; expr: smart_ptr<ast::Expression>*)

argument	argument type
self	*templates_boost::Template*
name	string const
expr	smart_ptr< *ast::Expression* >

Adds a rule to the template to replace a variable tag with an expression.

replaceArgumentWithList (*self: Template; name: string const; blka: array<smart_ptr<ast::Variable>> const*)

argument	argument type
self	*templates_boost::Template*
name	string const
blka	array< *VariablePtr* > const

Adds a rule to the template to replace a block argument with a list of variables.

replaceVariableWithList (*self: Template; name: string const; expr: array<smart_ptr<ast::Expression>> const*)

argument	argument type
self	*templates_boost::Template*
name	string const
expr	array< *ExpressionPtr* > const

Adds a rule to the template to replace a variable with an expression list.

replaceVariableWithList (*self: Template; name: string const; expr: dasvector`smart_ptr`Expression const*)

argument	argument type
self	*templates_boost::Template*
name	string const
expr	vector<smart_ptr<Expression>> const

Adds a rule to the template to replace a variable with an expression list.

renameField (*self: Template; name: string const; newName: string const*)

argument	argument type
self	*templates_boost::Template*
name	string const
newName	string const

Adds a rule to the template to rename any field lookup (., ?., as, is, etc)

renameField (*self: Template; name: string const; newName: das_string const*)

argument	argument type
self	*templates_boost::Template*
name	string const
newName	*builtin::das_string* const

Adds a rule to the template to rename any field lookup (., ?., as, is, etc)

replaceTypeWithTypeDecl (*self: Template; name: string const; expr: TypeDeclPtr*)

argument	argument type
self	*templates_boost::Template*
name	string const
expr	*TypeDeclPtr*

Adds a rule to the template to replace a type alias with another type alias, specified by type declaration.

renameCall (*self: Template; name: string const; newName: string const*)

argument	argument type
self	*templates_boost::Template*
name	string const
newName	string const

Adds a rule to the template to rename a call.

renameCall (*self: Template; name: string const; newName: das_string const*)

argument	argument type
self	*templates_boost::Template*
name	string const
newName	*builtin::das_string* const

Adds a rule to the template to rename a call.

make_expression_block (*exprs: array<smart_ptr<ast::Expression>>*)

make_expression_block returns smart_ptr< *ast::ExprBlock* >

argument	argument type
exprs	array< *ExpressionPtr* >

Create ExprBlock and move all expressions from expr to the list of the block.

make_expression_block (*exprs: dasvector`smart_ptr`Expression*)

make_expression_block returns smart_ptr< *ast::ExprBlock* >

argument	argument type
exprs	vector<smart_ptr<Expression>>

Create ExprBlock and move all expressions from expr to the list of the block.

add_type_ptr_ref (*a: TypeDeclPtr; flags: TypeDeclFlags*)

add_type_ptr_ref returns *TypeDeclPtr*

argument	argument type
a	*TypeDeclPtr*
flags	*TypeDeclFlags*

Implementation details for the reification. This adds any array to the rules.

add_type_ptr_ref (*st: StructurePtr; flags: TypeDeclFlags*)

add_type_ptr_ref returns *TypeDeclPtr*

argument	argument type
st	*StructurePtr*
flags	*TypeDeclFlags*

Implementation details for the reification. This adds any array to the rules.

apply_qmacro (*expr: smart_ptr<ast::Expression>; blk: block<(var rules:templates_boost::Template -const):void> const*)

apply_qmacro returns *ExpressionPtr*

argument	argument type
expr	smart_ptr< *ast::Expression* >
blk	block<(rules: *templates_boost::Template*):void> const

Implementation details for the expression reificaiton. This is a generat expression reification.

apply_qblock (*expr: smart_ptr<ast::Expression>; blk: block<(var rules:templates_boost::Template -const):void> const*)

apply_qblock returns *ExpressionPtr*

argument	argument type
expr	smart_ptr< *ast::Expression* >
blk	block<(rules: *templates_boost::Template*):void> const

Implementation details for the expression reificaiton. This is a block reification.

apply_qblock_to_array (*expr: smart_ptr<ast::Expression>; blk: block<(var rules:templates_boost::Template -const):void> const*)

apply_qblock_to_array returns array< *ExpressionPtr* >

argument	argument type
expr	smart_ptr< *ast::Expression* >
blk	block<(rules: *templates_boost::Template*):void> const

Implementation details for the expression reificaiton. This is a block reification.

apply_qblock_expr (*expr: smart_ptr<ast::Expression>; blk: block<(var rules:templates_boost::Template -const):void> const*)

apply_qblock_expr returns *ExpressionPtr*

argument	argument type
expr	smart_ptr< *ast::Expression* >
blk	block<(rules: *templates_boost::Template*):void> const

Implementation details for the expression reificaiton. This is a frist line of the block as expression reification.

apply_qtype (*expr: smart_ptr<ast::Expression>; blk: block<(var rules:templates_boost::Template -const):void> const*)

apply_qtype returns *TypeDeclPtr*

argument	argument type
expr	smart_ptr< *ast::Expression* >
blk	block<(rules: *templates_boost::Template*):void> const

Implementation details for the expression reificaiton. This is a type declaration reification.

expression_at (*expr: ExpressionPtr; at: LineInfo const*)

expression_at returns *ExpressionPtr*

argument	argument type
expr	*ExpressionPtr*
at	*rtti::LineInfo* const

Force expression location, than return it.

emplace_new (*arr: array<smart_ptr<ast::Expression>>; expr: ExpressionPtr*)

argument	argument type
arr	array< *ExpressionPtr* >
expr	*ExpressionPtr*

Unifies emplace and emplace_new for the array<VariablePtr>

emplace_new (*arr: array<smart_ptr<ast::Variable>>; expr: VariablePtr*)

argument	argument type
arr	array< *VariablePtr* >
expr	*VariablePtr*

Unifies emplace and emplace_new for the array<VariablePtr>

apply_qmacro_function (*fname: string const; expr: smart_ptr<ast::Expression>; blk: block<(var rules:templates_boost::Template -const):void> const*)

apply_qmacro_function returns *FunctionPtr*

argument	argument type
fname	string const
expr	smart_ptr< *ast::Expression* >
blk	block<(rules: *templates_boost::Template*):void> const

Implementation details for reification. This is a function generation reification.

apply_qmacro_method (*fname: string const; parent: StructurePtr; expr: smart_ptr<ast::Expression>; blk: block<(var rules:templates_boost::Template -const):void> const*)

apply_qmacro_method returns *FunctionPtr*

argument	argument type
fname	string const
parent	*StructurePtr*
expr	smart_ptr< *ast::Expression* >
blk	block<(rules: *templates_boost::Template*):void> const

Implementation details for reification. This is a class method function generation reification.

apply_qmacro_variable (*vname: string const; expr: smart_ptr<ast::Expression>; blk: block<(var rules:templates_boost::Template -const):void> const*)

apply_qmacro_variable returns *VariablePtr*

argument	argument type
vname	string const
expr	smart_ptr< *ast::Expression* >
blk	block<(rules: *templates_boost::Template*):void> const

Implementation details for reification. This is a variable generation reification.

add_structure_field (*cls: StructurePtr; name: string const; t: TypeDeclPtr; init: ExpressionPtr*)

add_structure_field returns int const

argument	argument type
cls	*StructurePtr*
name	string const
t	*TypeDeclPtr*
init	*ExpressionPtr*

Adds a field to the structure.

make_class (*name: string const; mod: rtti::Module? const*)

make_class returns smart_ptr< *ast::Structure* >

argument	argument type
name	string const
mod	*rtti::Module* ? const

Creates a class structure. Adds __rtti, __finalize fields.

make_class (*name: string const; baseClass: StructurePtr; mod: rtti::Module? const*)

make_class returns smart_ptr< *ast::Structure* >

argument	argument type
name	string const
baseClass	*StructurePtr*
mod	*rtti::Module* ? const

Creates a class structure. Adds __rtti, __finalize fields.

make_class (*name: string const; baseClass: ast::Structure? const; mod: rtti::Module? const*)

make_class returns smart_ptr< *ast::Structure* >

argument	argument type
name	string const
baseClass	*ast::Structure* ? const
mod	*rtti::Module* ? const

Creates a class structure. Adds __rtti, __finalize fields.

make_class_constructor (*cls: StructurePtr; ctor: FunctionPtr*)

make_class_constructor returns smart_ptr< *ast::Function* >

argument	argument type
cls	*StructurePtr*
ctor	*FunctionPtr*

Adds a class constructor from a constructor function.

modify_to_class_member (*cls: StructurePtr; fun: FunctionPtr; isExplicit: bool const; Constant: bool const*)

argument	argument type
cls	*StructurePtr*
fun	*FunctionPtr*
isExplicit	bool const
Constant	bool const

Modifies function to be a member of a particular class.

add_array_ptr_ref (*a: array<smart_ptr<auto(TT)>>*)

add_array_ptr_ref returns array<smart_ptr<TT>>

argument	argument type
a	array<smart_ptr<auto(TT)>>

Implementation details for the reification. This adds any array to the rules.

CHAPTER

THIRTYTHREE

BOOST PACKAGE FOR THE MISCELANIOUS MACRO MANIPULATIONS

Apply module implements miscellaneous infrastructure which simplifies writing of macros.

All functions and symbols are in “macro_boost” module, use require to get access to it.

```
require daslib/macro_boost
```

CapturedVariable

CapturedVariable fields are

variable	*ast::Variable* ?
expression	*ast::ExprVar* ?

Stored captured variable together with the *ExprVar* which uses it

33.1 Function annotations

MacroVerifyMacro

This macro implements *macro_verify* macro. It’s equivalent to a function call:

```
def macro_verify ( expr:bool; prog:ProgramPtr; at:LineInfo; message:string )
```

However, result will be substituted with:

```
if !expr
    macro_error( prog, at, message )
    return [[ExpressionPtr]]
```

33.2 Call macros

return_skip_lockcheck

this is similar to regular return <-, but it does not check for locks

33.3 Implementation details

- *macro_verify (expr:bool const;prog:smart_ptr<rtti::Program> const;at:rtti::LineInfo const;message:string const) : void*

macro_verify (*expr: bool const; prog: ProgramPtr; at: LineInfo const; message: string const*)

argument	argument type
expr	bool const
prog	*ProgramPtr*
at	*rtti::LineInfo* const
message	string const

Same as verify, only the check will produce macro error, followed by return [[ExpressionPtr]]

33.4 Uncategorized

capture_block (*expr: ExpressionPtr*)

capture_block returns array< *macro_boost::CapturedVariable* >

argument	argument type
expr	*ExpressionPtr*

Collect all captured variables in the expression.

collect_finally (*expr: ExpressionPtr*)

collect_finally returns array< *ast::ExprBlock* ?>

argument	argument type
expr	*ExpressionPtr*

Collect all finally blocks in the expression. Returns array of ExprBlock? with all the blocks which have *finally* section Does not go into 'make_block' expression, such as *lambda*, or 'block' expressions

collect_labels(*expr: ExpressionPtr*)

collect_labels returns array<int>

argument	argument type
expr	*ExpressionPtr*

Collect all labels in the expression. Returns array of integer with label indices Does not go into 'make_block' expression, such as *lambda*, or 'block' expressions

CHAPTER

THIRTYFOUR

IS_LOCAL_XXX AST HELPERS

The is_local module exposes collection of helper routines to establish locality of expression.

All functions and symbols are in "is_local" module, use require to get access to it.

```
require daslib/is_local
```

34.1 Scope checks

- *is_local_expr (expr:smart_ptr<ast::Expression> const) : bool const*
- *is_local_or_global_expr (expr:smart_ptr<ast::Expression> const) : bool const*
- *is_scope_expr (expr:smart_ptr<ast::Expression> const) : bool const*

is_local_expr(*expr: ExpressionPtr*)

is_local_expr returns bool const

argument	argument type
expr	*ExpressionPtr*

Returns true if the expression is local to the current scope.

is_local_or_global_expr(*expr: ExpressionPtr*)

is_local_or_global_expr returns bool const

argument	argument type
expr	*ExpressionPtr*

Returns true if expression is local the current scope or global scope.

is_scope_expr(*expr: ExpressionPtr*)

is_scope_expr returns bool const

argument	argument type
expr	*ExpressionPtr*

Returns true if the expression is a scoped expression, i.e. eventually points to a variable.

34.2 Uncategorized

is_shared_expr (*expr: ExpressionPtr*)

is_shared_expr returns bool const

argument	argument type
expr	*ExpressionPtr*

Returns true if the expression is local to the current scope.

CHAPTER

THIRTYFIVE

SAFE_ADDR MACRO

The safe_addr module implements safe_addr pattern, which returns temporary address of local expression.

All functions and symbols are in "safe_addr" module, use require to get access to it.

```
require daslib/safe_addr
```

35.1 Function annotations

SafeAddrMacro

This macro reports an error if safe_addr is attempted on the object, which is not local to the scope. I.e. if the object can *expire* while in scope, with delete, garbage collection, or on the C++ side.

SharedAddrMacro

|function_annotation-safe_addr-SharedAddrMacro|

35.2 Safe temporary address

- *safe_addr (x:auto(T)& ==const -const) : T -&?#*
- *safe_addr (x:auto(T) const& ==const) : T -&? const#*
- *shared_addr (tab:table<auto(KEY);auto(VAL)> const;k:KEY const) : auto*
- *shared_addr (val:auto(VALUE) const&) : auto*

safe_addr (*x: auto(T)& ==const*)

safe_addr returns T?#

argument	argument type
x	auto(T)&!

returns temporary pointer to the given expression

safe_addr (*x: auto(T) const& ==const*) safe_addr returns T? const#

argument	argument type
x	auto(T) const&!

returns temporary pointer to the given expression

shared_addr (*tab: table<auto(KEY);auto(VAL)> const; k: KEY const*)

shared_addr returns auto

argument	argument type
tab	table<auto(KEY);auto(VAL)> const
k	KEY const

returns address of the given shared variable. it's safe because shared variables never go out of scope

shared_addr (*val: auto(VALUE) const&*)

shared_addr returns auto

argument	argument type
val	auto(VALUE) const&

returns address of the given shared variable. it's safe because shared variables never go out of scope

35.3 Temporary pointers

- *temp_ptr (x:auto(T)? const implicit ==const) : T? const#*
- *temp_ptr (x:auto(T)? implicit ==const -const) : T?#*

temp_ptr (*x: auto(T)? const implicit ==const*)

temp_ptr returns T? const#

argument	argument type
x	auto(T)? const implicit!

returns temporary pointer from a given pointer

temp_ptr (*x: auto(T)? implicit ==const*) temp_ptr returns T?#

argument	argument type
x	auto(T)? implicit!

returns temporary pointer from a given pointer

CHAPTER

THIRTYSIX

STATIC_LET MACRO

The static_let module implements static_let pattern, which allows declaration of private global variables which are local to a scope.

All functions and symbols are in "static_let" module, use require to get access to it.

```
require daslib/static_let
```

36.1 Function annotations

StaticLetMacro

This macro implements the *static_let* and *static_let_finalize* functions.

36.2 Static variable declarations

- *static_let (blk:block<> const) : void*
- *static_let_finalize (blk:block<> const) : void*

static_let (*blk: block<> const*)

argument	argument type
blk	block<> const

Given a scope with the variable declarations, this function will make those variables global. Variable will be renamed under the hood, and all local access to it will be renamed as well.

static_let_finalize (*blk: block<> const*)

argument	argument type
blk	block<> const

This is very similar to regular static_let, but additionally the variable will be deleted on the context shutdown.

CHAPTER

THIRTYSEVEN

LPIPE MACRO

The lpipe module implements lpipe pattern, which allows piping blocks and expressions onto the previous line.

All functions and symbols are in "lpipe" module, use require to get access to it.

```
require daslib/lpipe
```

37.1 Call macros

lpipe

This macro will implement the lpipe function. It allows piping blocks the previous line call. For example:

```
def take2(a,b:block)
    invoke(a)
    invoke(b)
...
take2 <|
    print("block1\n")
lpipe <|    // this block will pipe into take2
    print("block2\n")
```

CHAPTER

THIRTYEIGHT

BOOST PACKAGE FOR ARRAY MANIPULATION

The array_boost module implements collection of array manipulation routines.

All functions and symbols are in "array_boost" module, use require to get access to it.

```
require daslib/array_boost
```

38.1 Temporary arrays

- *temp_array (arr:auto implicit ==const -const) : auto*
- *temp_array (arr:auto const implicit ==const) : auto*
- *temp_array (data:auto? ==const -const;lenA:int const;a:auto(TT) const) : array<TT -const -#>*
- *temp_array (data:auto? const ==const;lenA:int const;a:auto(TT) const) : array<TT -const -#> const*

temp_array (*arr: auto implicit ==const*)

temp_array returns auto

Warning: This is unsafe operation.

argument	argument type
arr	auto implicit!

Creates temporary array from the given object. Important requirements are:

- object memory is linear
- each element follows the next one directly, with the stride equal to size of the element
- object memory does not change within the lifetime of the returned array

temp_array (*arr: auto const implicit ==const*)

temp_array returns auto

Warning: This is unsafe operation.

argument	argument type
arr	auto const implicit!

Creates temporary array from the given object. Important requirements are:

- object memory is linear
- each element follows the next one directly, with the stride equal to size of the element
- object memory does not change within the lifetime of the returned array

`temp_array` (*data: auto? ==const; lenA: int const; a: auto(TT) const*)

temp_array returns array<TT>

> **Warning:** This is unsafe operation.

argument	argument type
data	auto?!
lenA	int const
a	auto(TT) const

Creates temporary array from the given object. Important requirements are:

- object memory is linear
- each element follows the next one directly, with the stride equal to size of the element
- object memory does not change within the lifetime of the returned array

`temp_array` (*data: auto? const ==const; lenA: int const; a: auto(TT) const*)

temp_array returns array<TT> const

> **Warning:** This is unsafe operation.

argument	argument type
data	auto? const!
lenA	int const
a	auto(TT) const

Creates temporary array from the given object. Important requirements are:

- object memory is linear

- each element follows the next one directly, with the stride equal to size of the element
- object memory does not change within the lifetime of the returned array

38.2 Empty check

- *empty (v:auto(VecT) const) : auto*

empty (*v: auto(VecT) const*)

empty returns auto

argument	argument type
v	auto(VecT) const

returns true if 'v' has 0 elements. this also implies that *length(v)* is defined.

38.3 Uncategorized

array_view (*bytes: array<auto(TT)> const ==const; offset: int const; length: int const; blk: block<(view:array<TT> const#):void> const*)

array_view returns auto

argument	argument type
bytes	array<auto(TT)> const!
offset	int const
length	int const
blk	block<(view:array<TT> const#):void> const

creates a view of the array, which is a temporary array that is valid only within the block

array_view (*bytes: array<auto(TT)> ==const; offset: int const; length: int const; blk: block<(var view:array<TT># -const):void> const*)

array_view returns auto

argument	argument type
bytes	array<auto(TT)>!
offset	int const
length	int const
blk	block<(view:array<TT>#):void> const

creates a view of the array, which is a temporary array that is valid only within the block

CHAPTER

THIRTYNINE

GENERAL PRUPOSE SERIALIZATION

The archive module implements general purpose serialization infrastructure.

All functions and symbols are in "archive" module, use require to get access to it.

```
require daslib/archive
```

To correctly support serialization of the specific type, you need to define and implement *serialize* method for it. For example this is how DECS implements component serialization:

```
def public serialize ( var arch:Archive; var src:Component )
    arch |> serialize(src.name)
    arch |> serialize(src.hash)
    arch |> serialize(src.stride)
    arch |> serialize(src.info)
    invoke(src.info.serializer, arch, src.data)
```

Archive

Archive fields are

version	uint
reading	bool
stream	*archive::Serializer* ?

Archive is a combination of serialization stream, and state (version, and reading status).

39.1 Classes

Serializer

Base class for serializers.

it defines as follows

`Serializer.`**`write`**`(`*self: Serializer; bytes: void? const implicit; size: int const*`)`

write returns bool

argument	argument type
self	*archive::Serializer*
bytes	void? const implicit
size	int const

Write binary data to stream.

`Serializer.`**`read`**(*self: Serializer; bytes: void? const implicit; size: int const*)

read returns bool

argument	argument type
self	*archive::Serializer*
bytes	void? const implicit
size	int const

Read binary data from stream.

`Serializer.`**`error`**(*self: Serializer; code: string const*)

argument	argument type
self	*archive::Serializer*
code	string const

Report error to the archive

`Serializer.`**`OK`**(*self: Serializer*)

OK returns bool

Return status of the archive

`MemSerializer : Serializer`

This serializer stores data in memory (in the array<uint8>)

it defines as follows

> data : array<uint8>
> readOffset : int
> lastError : string

`MemSerializer.`**`write`**(*self: Serializer; bytes: void? const implicit; size: int const*)

write returns bool

argument	argument type
self	*archive::Serializer*
bytes	void? const implicit
size	int const

Appends bytes at the end of the data.

`MemSerializer.`**`read`**(*self: Serializer; bytes: void? const implicit; size: int const*)

read returns bool

argument	argument type
self	*archive::Serializer*
bytes	void? const implicit
size	int const

Reads bytes from data, advances the reading position.

`MemSerializer.`**`error`**(*self: Serializer; code: string const*)

argument	argument type
self	*archive::Serializer*
code	string const

Sets the last error code.

`MemSerializer.`**`OK`**(*self: Serializer*)

OK returns bool

Implements 'OK' method, which returns true if the serializer is in a valid state.

`MemSerializer.`**`extractData`**(*self: MemSerializer*)

extractData returns array<uint8>

Extract the data from the serializer.

`MemSerializer.`**`getCopyOfData`**(*self: MemSerializer*)

getCopyOfData returns array<uint8>

Returns copy of the data from the seiralizer.

`MemSerializer.`**`getLastError`**(*self: MemSerializer*)

getLastError returns string

Returns last serialization error.

39.2 Serialization

- *serialize (arch:archive::Archive -const;value:math::float3x3 -const) : void*
- *serialize (arch:archive::Archive -const;value:math::float3x4 -const) : void*
- *serialize (arch:archive::Archive -const;value:math::float4x4 -const) : void*
- *serialize (arch:archive::Archive -const;value:string& -const) : void*
- *serialize_raw (arch:archive::Archive -const;value:auto(TT)& -const) : auto*
- *read_raw (arch:archive::Archive -const;value:auto(TT)& -const) : auto*
- *write_raw (arch:archive::Archive -const;value:auto(TT)& -const) : auto*
- *serialize (arch:archive::Archive -const;value:auto(TT)& -const) : auto*
- *serialize (arch:archive::Archive -const;value:auto(TT)& -const) : auto*
- *serialize (arch:archive::Archive -const;value:auto(TT)& -const) : auto*
- *serialize (arch:archive::Archive -const;value:auto(TT)& -const) : auto*
- *serialize (arch:archive::Archive -const;value:auto(TT)& -const) : auto*
- *serialize (arch:archive::Archive -const;value:auto(TT)[] -const) : auto*
- *serialize (arch:archive::Archive -const;value:array<auto(TT)> -const) : auto*
- *serialize (arch:archive::Archive -const;value:table<auto(KT);auto(VT)> -const) : auto*
- *serialize (arch:archive::Archive -const;value:auto(TT)? -const) : auto*

serialize(*arch: Archive; value: float3x3*)

argument	argument type
arch	*archive::Archive*
value	*math::float3x3*

Serializes structured data, based on the *value* type.

serialize(*arch: Archive; value: float3x4*)

argument	argument type
arch	*archive::Archive*
value	*math::float3x4*

Serializes structured data, based on the *value* type.

serialize(*arch: Archive; value: float4x4*)

argument	argument type
arch	*archive::Archive*
value	*math::float4x4*

Serializes structured data, based on the *value* type.

serialize (*arch: Archive; value: string&*)

argument	argument type
arch	*archive::Archive*
value	string&

Serializes structured data, based on the *value* type.

serialize_raw (*arch: Archive; value: auto(TT)&*)

serialize_raw returns auto

argument	argument type
arch	*archive::Archive*
value	auto(TT)&

Serialize raw data (straight up bytes for raw pod)

read_raw (*arch: Archive; value: auto(TT)&*)

read_raw returns auto

argument	argument type
arch	*archive::Archive*
value	auto(TT)&

Read raw data (straight up bytes for raw pod)

write_raw (*arch: Archive; value: auto(TT)&*)

write_raw returns auto

argument	argument type
arch	*archive::Archive*
value	auto(TT)&

Write raw data (straight up bytes for raw pod)

serialize (*arch: Archive; value: auto(TT)&*)

serialize returns auto

argument	argument type
arch	*archive::Archive*
value	auto(TT)&

Serializes structured data, based on the *value* type.

serialize (*arch: Archive; value: auto(TT)&*)

serialize returns auto

argument	argument type
arch	*archive::Archive*
value	auto(TT)&

Serializes structured data, based on the *value* type.

serialize (*arch: Archive; value: auto(TT)&*)

serialize returns auto

argument	argument type
arch	*archive::Archive*
value	auto(TT)&

Serializes structured data, based on the *value* type.

serialize (*arch: Archive; value: auto(TT)&*)

serialize returns auto

argument	argument type
arch	*archive::Archive*
value	auto(TT)&

Serializes structured data, based on the *value* type.

serialize (*arch: Archive; value: auto(TT)&*)

serialize returns auto

argument	argument type
arch	*archive::Archive*
value	auto(TT)&

Serializes structured data, based on the *value* type.

serialize (*arch: Archive; value: auto(TT)[]*)

serialize returns auto

argument	argument type
arch	*archive::Archive*
value	auto(TT)[-1]

Serializes structured data, based on the *value* type.

serialize (*arch: Archive; value: array<auto(TT)>*)

serialize returns auto

argument	argument type
arch	*archive::Archive*
value	array<auto(TT)>

Serializes structured data, based on the *value* type.

serialize (*arch: Archive; value: table<auto(KT);auto(VT)>*)

serialize returns auto

argument	argument type
arch	*archive::Archive*
value	table<auto(KT);auto(VT)>

Serializes structured data, based on the *value* type.

serialize (*arch: Archive; value: auto(TT)?*)

serialize returns auto

argument	argument type
arch	*archive::Archive*
value	auto(TT)?

Serializes structured data, based on the *value* type.

39.3 Memory archive

- *mem_archive_save (t:auto& -const) : auto*
- *mem_archive_load (data:array<uint8> -const;t:auto& -const;canfail:bool const) : bool*

mem_archive_save (*t: auto&*)

mem_archive_save returns auto

argument	argument type
t	auto&

Saves the object to a memory archive. Result is array<uint8> with the serialized data.

mem_archive_load (*data: array<uint8>; t: auto&; canfail: bool const*)

mem_archive_load returns bool

argument	argument type
data	array<uint8>
t	auto&
canfail	bool const

Loads the object from a memory archive. *data* is the array<uint8> with the serialized data, returned from *mem_archive_save*.

CHAPTER

FORTY

LOOP UNROLLING

The unroll module implements loop unrolling infrastructure.

All functions and symbols are in "unroll" module, use require to get access to it.

```
require daslib/unroll
```

40.1 Function annotations

UnrollMacro

This macro implements loop unrolling in the form of *unroll* function. Unroll function expects block with the single for loop in it. Moveover only range for is supported, and only with the fixed range. For example::

```
var n : float4[9]
unroll <|   // contents of the loop will be replaced with 9 image load instructions.
    for i in range(9)
        n[i] = imageLoad(c_bloom_htex, xy + int2(0,i-4))
```

40.2 Unrolling

- *unroll (blk:block<> const) : void*

unroll (*blk: block<> const*)

argument	argument type
blk	block<> const

Unrolls the for loop (with fixed range)

CHAPTER

FORTYONE

ASSERT ONCE

The assert_once module implements single-time assertion infrastructure.

All functions and symbols are in “assert_once” module, use require to get access to it.

```
require daslib/assert_once
```

41.1 Function annotations

AssertOnceMacro

This macro convert assert_once(expr,message) to the following code:

```
var __assert_once_I = true  // this is a global variable
if __assert_once_I && !expr
    __assert_once_I = false
    assert(false,message)
```

41.2 Assertion

- *assert_once (expr:bool const;message:string const) : void*

assert_once (*expr: bool const; message: string const*)

argument	argument type
expr	bool const
message	string const

Same as assert, only the check will be not be repeated after the asseretion failed the first time.

CHAPTER

FORTYTWO

DECS, AST BLOCK TO LOOP

The ast_block_to_loop module implements block to loop conversion as part of the DECS infrastructure.

All functions and symbols are in "ast_block_to_loop" module, use require to get access to it.

```
require daslib/ast_block_to_loop
```

42.1 Block to loop conversion

- *convert_block_to_loop (blk:smart_ptr<ast::Expression> -const;failOnReturn:bool const;replaceReturnWithContinue:bool const;requireContinueCond:bool const) : void*

convert_block_to_loop(*blk: smart_ptr<ast::Expression>; failOnReturn: bool const; replaceReturnWithContinue: bool const; requireContinueCond: bool const*)

argument	argument type
blk	smart_ptr< *ast::Expression* >
failOnReturn	bool const
replaceReturnWithContinue	bool const
requireContinueCond	bool const

Converts closure block to loop. If *failOnReturn* is true, then returns are not allowed inside the block. If *replaceReturnWithContinue* is true, then *return cond;* are replaced with *if cond; continue;*. If *requireContinueCond* is false, then *return;* is replaced with *continue;*, otherwise it is an error.

CHAPTER FORTYTHREE

AST TYPE USSAGE COLLECTION

The ast_used module implements type collecting infrasturcture. It allows to determine, if enumeration and structure types are used in the code.

All functions and symbols are in “ast_used” module, use require to get access to it.

```
require daslib/ast_used
```

OnlyUsedTypes

OnlyUsedTypes fields are

st	table< *ast::Structure* ?;bool>
en	table< *ast::Enumeration* ?;bool>

Collection of all structure and enumeration types that are used in the AST.

43.1 Collecting type information

- *collect_used_types (vfun:array<ast::Function?> const;vvar:array<ast::Variable?> const;blk:block<(usedTypes:ast_used::OnlyUsedTypes const):void> const) : void*

collect_used_types (*vfun: array<ast::Function?> const; vvar: array<ast::Variable?> const; blk: block<(usedTypes:ast_used::OnlyUsedTypes const):void> const*)

argument	argument type
vfun	array< *ast::Function* ?> const
vvar	array< *ast::Variable* ?> const
blk	block<(usedTypes: *ast_used::OnlyUsedTypes* const):void> const

Goes through list of functions *vfun* and variables *vvar* and collects list of which enumeration and structure types are used in them. Calls *blk* with said list.

CHAPTER

FORTYFOUR

CONSTANT EXPRESSION CHECKER AND SUBSTITUTION

The constant_expression module implements *constant expression* function argument check, as well as argument substitution.

All functions and symbols are in “constexpr” module, use require to get access to it.

```
require daslib/constant_expression
```

44.1 Function annotations

constexpr

This macro implements a constexpr function argument checker. Given list of arguments to verify, it will fail for every one where non-constant expression is passed. For example:

```
[constexpr (a)]
def foo ( t:string; a : int )
    print("{t} = {a}\n")
var BOO = 13
[export]
def main
    foo("blah", 1)
    foo("ouch", BOO)    // comilation error: `a is not a constexpr, BOO`
```

constant_expression

This function annotation implments constant expression folding for the given arguments. When argument is specified in the annotation, and is passed as a contstant expression, custom version of the function is generated, and an argument is substituted with a constant value. This allows using of static_if expression on the said arguments, as well as other optimizations. For example:

```
[constant_expression(constString)]
def take_const_arg(constString:string)
    print("constant string is = {constString}\n")   // note - constString here is not␣
↪an argument
```

44.2 Macro helpers

- *isConstantExpression (expr:smart_ptr<ast::Expression> const) : bool*

isConstantExpression(*expr: ExpressionPtr*)

isConstantExpression returns bool

argument	argument type
expr	*ExpressionPtr*

This macro function retrusn true if the expression is a constant expression

CHAPTER

FORTYFIVE

BOOST PACKAGE FOR THE BUILTIN SORT

The sort_boost module implements additional infrastructure for the sorting routines.

All functions and symbols are in "sort_boost" module, use require to get access to it.

```
require daslib/sort_boost
```

45.1 Call macros

qsort

Implements *qsort* macro. I'ts *qsort(value,block)*. For the regular array<> or dim it's replaced with *sort(value,block)*. For the handled types like das`vector its replaced with *sort(temp_array(value),block)*.

CHAPTER

FORTYSIX

ENUMERATION TRAITS

The enum_trait module implements typeinfo traits for the enumerations.

All functions and symbols are in "enum_trait" module, use require to get access to it.

```
require daslib/enum_trait
```

46.1 Typeinfo macros

enum_names

Implements typeinfo(enum_names EnumOrEnumType) which returns array of strings with enumValue names.

enum_length

Implements typeinfo(enum_length EnumOrEnumType) which returns total number of elements in enumeration.

CHAPTER FORTYSEVEN

C++ BINDINGS GENERATOR

The cpp_bind module implements generation of C++ bindings for the Daslang interfaces.

All functions and symbols are in "cpp_bind" module, use require to get access to it.

```
require daslib/cpp_bind
```

For example, from tutorial04.das

```
require fio
require ast
require daslib/cpp_bind
[init]
def generate_cpp_bindings
    let root = get_das_root() + "/examples/tutorial/"
    fopen(root + "tutorial04_gen.inc","wb") <| $ ( cpp_file )
        log_cpp_class_adapter(cpp_file, "TutorialBaseClass", typeinfo(ast_typedecl␣
↪type<TutorialBaseClass>))
```

47.1 Generation of bindings

- *log_cpp_class_adapter (cpp_file:fio::FILE const? const;name:string const;cinfo:smart_ptr<ast::TypeDecl> const) : void*

log_cpp_class_adapter(*cpp_file: file; name: string const; cinfo: TypeDeclPtr*)

argument	argument type
cpp_file	*file*
name	string const
cinfo	*TypeDeclPtr*

Generates C++ class adapter for the Daslang class. Intended use:

```
log_cpp_class_adapter(cppFileNameDotInc, "DaslangClassName", typeinfo(ast_typedecl␣
↪type<DaslangClassName>))
```

CHAPTER

FORTYEIGHT

DECS, DASLANG ENTITY COMPONENT SYSTEM

The DECS module implements low level functionality of Daslang entity component system.

All functions and symbols are in "decs" module, use require to get access to it.

```
require daslib/decs
```

Under normal circumstances, the module is not used without the boost package:

```
require daslib/desc_boost
```

48.1 Type aliases

ComponentHash = uint64

Hash value of the ECS component type

TypeHash = uint64

Hash value of the individual type

DeferEval = lambda<(var act:DeferAction -const):void>

Lambda which holds deferred action. Typically creation of destruction of an entity.

ComponentMap = array<decs::ComponentValue>

Table of component values for individual entity.

PassFunction = function<void>

One of the callbacks which form individual pass.

CTypeInfo

CTypeInfo fields are

basicType	*rtti::Type*
mangledName	string
fullName	string
hash	*TypeHash*
size	uint
eraser	function<(arr:array<uint8>):void>
clonner	function<(dst:array<uint8>;src:array<uint8> const):void>
serializer	function<(arch: *archive::Archive* ;arr:array<uint8>):void>
dumper	function<(elem:void? const):string>
mkTypeInfo	function<>
gc	function<(src:array<uint8>):lambda<>>

Type information for the individual component subtype. Consists of type name and collection of type-specific routines to control type values during its lifetime, serialization, etc.

Component

Component fields are

name	string
hash	*ComponentHash*
stride	int
data	array<uint8>
info	*decs::CTypeInfo*
gc_dummy	lambda<>

Single ECS component. Contains component name, data, and data layout.

EntityId

EntityId fields are

id	uint
generation	int

Unique identifier of the entity. Consists of id (index in the data array) and generation.

Archetype

Archetype fields are

hash	*ComponentHash*
components	array< *decs::Component* >
size	int
eidIndex	int

ECS archetype. Archetype is unique combination of components.

ComponentValue

ComponentValue fields are

name	string
info	*decs::CTypeInfo*
data	float4[4]

Value of the component during creation or transformation.

EcsRequestPos

EcsRequestPos fields are

file	string
line	uint

Location of the ECS request in the code (source file and line number).

EcsRequest

EcsRequest fields are

hash	*ComponentHash*
req	array<string>
reqn	array<string>
archetypes	array<int>
at	*decs::EcsRequestPos*

Individual ESC requests. Contains list of required components, list of components which are required to be absent. Caches list of archetypes, which match the request.

DecsState

DecsState fields are

archetypeLookup	table< *ComponentHash* ;int>
allArchetypes	array< *decs::Archetype* >
entityFreeList	array< *decs::EntityId* >
entityLookup	array<tuple<generation:int;archetype: *ComponentHash* ;index:int>>
componentTypeCheck	table<string; *decs::CTypeInfo* >
ecsQueries	array< *decs::EcsRequest* >
queryLookup	table< *ComponentHash* ;int>

Entire state of the ECS system. Conntains archtypes, entities and entity free-list, entity lokup table, all archetypes and archetype lookups, etc.

DecsPass

DecsPass fields are

name	string
calls	array< *PassFunction* >

Individual pass of the update of the ECS system. Contains pass name and list of all pass calblacks.

48.2 Comparison and access

- *== (a:decs::EntityId const implicit;b:decs::EntityId const implicit) : bool*
- *!= (a:decs::EntityId const implicit;b:decs::EntityId const implicit) : bool*
- *. (cmp:array<decs::ComponentValue> -const;name:string const) : decs::ComponentValue&*

operator == (*a: EntityId const implicit; b: EntityId const implicit*)

== returns bool

argument	argument type
a	*decs::EntityId* const implicit
b	*decs::EntityId* const implicit

Equality operator for entity IDs.

operator !=(*a: EntityId const implicit; b: EntityId const implicit*)

!= returns bool

argument	argument type
a	*decs::EntityId* const implicit
b	*decs::EntityId* const implicit

Inequality operator for entity IDs.

operator .(*cmp: ComponentMap; name: string const*)

. returns *decs::ComponentValue* &

argument	argument type
cmp	*ComponentMap*
name	string const

Access to component value by name. For example:

```
create_entity <| @ ( eid, cmp )
    cmp.pos := float3(i)     // same as cmp |> set("pos",float3(i))
```

48.3 Access (get/set/clone)

- *clone (cv:decs::ComponentValue -const;val:decs::EntityId const) : void*
- *clone (cv:decs::ComponentValue -const;val:bool const) : void*
- *clone (cv:decs::ComponentValue -const;val:range const) : void*
- *clone (cv:decs::ComponentValue -const;val:urange const) : void*
- *clone (cv:decs::ComponentValue -const;val:range64 const) : void*
- *clone (cv:decs::ComponentValue -const;val:urange64 const) : void*
- *clone (cv:decs::ComponentValue -const;val:string const) : void*
- *clone (cv:decs::ComponentValue -const;val:int const) : void*
- *clone (cv:decs::ComponentValue -const;val:int8 const) : void*
- *clone (cv:decs::ComponentValue -const;val:int16 const) : void*
- *clone (cv:decs::ComponentValue -const;val:int64 const) : void*
- *clone (cv:decs::ComponentValue -const;val:int2 const) : void*
- *clone (cv:decs::ComponentValue -const;val:int3 const) : void*
- *clone (cv:decs::ComponentValue -const;val:int4 const) : void*

- *clone (cv:decs::ComponentValue -const;val:uint const) : void*
- *clone (cv:decs::ComponentValue -const;val:uint8 const) : void*
- *clone (cv:decs::ComponentValue -const;val:uint16 const) : void*
- *clone (cv:decs::ComponentValue -const;val:uint64 const) : void*
- *clone (cv:decs::ComponentValue -const;val:uint2 const) : void*
- *clone (cv:decs::ComponentValue -const;val:uint3 const) : void*
- *clone (cv:decs::ComponentValue -const;val:uint4 const) : void*
- *clone (cv:decs::ComponentValue -const;val:float const) : void*
- *clone (cv:decs::ComponentValue -const;val:float2 const) : void*
- *clone (cv:decs::ComponentValue -const;val:float3 const) : void*
- *clone (cv:decs::ComponentValue -const;val:float4 const) : void*
- *clone (cv:decs::ComponentValue -const;val:math::float3x3 const) : void*
- *clone (cv:decs::ComponentValue -const;val:math::float3x4 const) : void*
- *clone (cv:decs::ComponentValue -const;val:math::float4x4 const) : void*
- *clone (cv:decs::ComponentValue -const;val:double const) : void*
- *clone (dst:decs::Component -const;src:decs::Component const) : void*
- *has (arch:decs::Archetype const;name:string const) : bool*
- *has (cmp:array<decs::ComponentValue> -const;name:string const) : bool*
- *remove (cmp:array<decs::ComponentValue> -const;name:string const) : void*
- *set (cv:decs::ComponentValue -const;val:auto const) : auto*
- *get (arch:decs::Archetype const;name:string const;value:auto(TT) const) : auto*
- *get (cmp:array<decs::ComponentValue> -const;name:string const;value:auto(TT) -const) : auto*
- *set (cmp:array<decs::ComponentValue> -const;name:string const;value:auto(TT) const) : auto*

clone(*cv: ComponentValue; val: EntityId const*)

argument	argument type
cv	*decs::ComponentValue*
val	*decs::EntityId* const

Clones component value.

clone(*cv: ComponentValue; val: bool const*)

argument	argument type
cv	*decs::ComponentValue*
val	bool const

Clones component value.

clone (*cv: ComponentValue; val: range const*)

argument	argument type
cv	*decs::ComponentValue*
val	range const

Clones component value.

clone (*cv: ComponentValue; val: urange const*)

argument	argument type
cv	*decs::ComponentValue*
val	urange const

Clones component value.

clone (*cv: ComponentValue; val: range64 const*)

argument	argument type
cv	*decs::ComponentValue*
val	range64 const

Clones component value.

clone (*cv: ComponentValue; val: urange64 const*)

argument	argument type
cv	*decs::ComponentValue*
val	urange64 const

Clones component value.

clone (*cv: ComponentValue; val: string const*)

argument	argument type
cv	*decs::ComponentValue*
val	string const

Clones component value.

clone (*cv: ComponentValue; val: int const*)

argument	argument type
cv	*decs::ComponentValue*
val	int const

Clones component value.

clone (*cv: ComponentValue; val: int8 const*)

argument	argument type
cv	*decs::ComponentValue*
val	int8 const

Clones component value.

clone (*cv: ComponentValue; val: int16 const*)

argument	argument type
cv	*decs::ComponentValue*
val	int16 const

Clones component value.

clone (*cv: ComponentValue; val: int64 const*)

argument	argument type
cv	*decs::ComponentValue*
val	int64 const

Clones component value.

clone (*cv: ComponentValue; val: int2 const*)

argument	argument type
cv	*decs::ComponentValue*
val	int2 const

Clones component value.

clone (*cv: ComponentValue; val: int3 const*)

argument	argument type
cv	*decs::ComponentValue*
val	int3 const

Clones component value.

clone (*cv: ComponentValue; val: int4 const*)

argument	argument type
cv	*decs::ComponentValue*
val	int4 const

Clones component value.

clone (*cv: ComponentValue; val: uint const*)

argument	argument type
cv	*decs::ComponentValue*
val	uint const

Clones component value.

clone (*cv: ComponentValue; val: uint8 const*)

argument	argument type
cv	*decs::ComponentValue*
val	uint8 const

Clones component value.

clone (*cv: ComponentValue; val: uint16 const*)

argument	argument type
cv	*decs::ComponentValue*
val	uint16 const

Clones component value.

clone (*cv: ComponentValue; val: uint64 const*)

argument	argument type
cv	*decs::ComponentValue*
val	uint64 const

Clones component value.

clone (*cv: ComponentValue; val: uint2 const*)

argument	argument type
cv	*decs::ComponentValue*
val	uint2 const

Clones component value.

clone (*cv: ComponentValue; val: uint3 const*)

argument	argument type
cv	*decs::ComponentValue*
val	uint3 const

Clones component value.

clone (*cv: ComponentValue; val: uint4 const*)

argument	argument type
cv	*decs::ComponentValue*
val	uint4 const

Clones component value.

clone (*cv: ComponentValue; val: float const*)

argument	argument type
cv	*decs::ComponentValue*
val	float const

Clones component value.

clone (*cv: ComponentValue; val: float2 const*)

argument	argument type
cv	*decs::ComponentValue*
val	float2 const

Clones component value.

clone (*cv: ComponentValue; val: float3 const*)

argument	argument type
cv	*decs::ComponentValue*
val	float3 const

Clones component value.

clone (*cv: ComponentValue; val: float4 const*)

argument	argument type
cv	*decs::ComponentValue*
val	float4 const

Clones component value.

clone (*cv: ComponentValue; val: float3x3 const*)

argument	argument type
cv	*decs::ComponentValue*
val	*math::float3x3* const

Clones component value.

clone (*cv: ComponentValue; val: float3x4 const*)

argument	argument type
cv	*decs::ComponentValue*
val	*math::float3x4* const

Clones component value.

clone (*cv: ComponentValue; val: float4x4 const*)

argument	argument type
cv	*decs::ComponentValue*
val	*math::float4x4* const

Clones component value.

clone (*cv: ComponentValue; val: double const*)

argument	argument type
cv	*decs::ComponentValue*
val	double const

Clones component value.

clone (*dst: Component; src: Component const*)

argument	argument type
dst	*decs::Component*
src	*decs::Component* const

Clones component value.

has (*arch: Archetype const; name: string const*)

has returns bool

argument	argument type
arch	*decs::Archetype* const
name	string const

Returns true if object has specified subobjec.

has (*cmp: ComponentMap; name: string const*)

has returns bool

argument	argument type
cmp	*ComponentMap*
name	string const

Returns true if object has specified subobjec.

remove (*cmp: ComponentMap; name: string const*)

argument	argument type
cmp	*ComponentMap*
name	string const

Removes speicified value from the component map.

set (*cv: ComponentValue; val: auto const*)

set returns auto

argument	argument type
cv	*decs::ComponentValue*
val	auto const

Set component value specified by name and type. If value already exists, it is overwritten. If already existing value type is not the same - panic.

get (*arch: Archetype const; name: string const; value: auto(TT) const*)

get returns auto

argument	argument type
arch	*decs::Archetype* const
name	string const
value	auto(TT) const

Gets component value specified by name and type. Will panic if name matches but type does not.

get (*cmp: ComponentMap; name: string const; value: auto(TT)*)

get returns auto

argument	argument type
cmp	*ComponentMap*
name	string const
value	auto(TT)

Gets component value specified by name and type. Will panic if name matches but type does not.

set (*cmp: ComponentMap; name: string const; value: auto(TT) const*)

set returns auto

argument	argument type
cmp	*ComponentMap*
name	string const
value	auto(TT) const

Set component value specified by name and type. If value already exists, it is overwritten. If already existing value type is not the same - panic.

48.4 Deubg and serialization

- *describe (info:decs::CTypeInfo const) : string*
- *serialize (arch:archive::Archive -const;src:decs::Component -const) : void*
- *finalize (cmp:decs::Component -const) : void*
- *debug_dump () : void*

describe (*info: CTypeInfo const*)

describe returns string

argument	argument type
info	*decs::CTypeInfo* const

Returns textual description of the type.

serialize (*arch: Archive; src: Component*)

argument	argument type
arch	*archive::Archive*
src	*decs::Component*

Serializes component value.

finalize (*cmp: Component*)

argument	argument type
cmp	*decs::Component*

Deletes component.

debug_dump ()

Prints out state of the ECS system.

48.5 Stages

- *register_decs_stage_call (name:string const;pcall:function<void> const) : void*
- *decs_stage (name:string const) : void*
- *commit () : void*

register_decs_stage_call (*name: string const; pcall: PassFunction*)

argument	argument type
name	string const
pcall	*PassFunction*

Registration of a single pass callback. This is a low-level function, used by decs_boost macros.

decs_stage (*name: string const*)

argument	argument type
name	string const

Invokes specific ECS pass. *commit* is called before and after the invocation.

commit ()

Finishes all deferred actions.

48.6 Deferred actions

- *update_entity (entityid:decs::EntityId const implicit;blk:lambda<(eid:decs::EntityId const;var cmp:array<decs::ComponentValue> -const):void> -const) : void*
- *create_entity (blk:lambda<(eid:decs::EntityId const;var cmp:array<decs::ComponentValue> -const):void> -const) : decs::EntityId*
- *delete_entity (entityid:decs::EntityId const implicit) : void*

update_entity (*entityid: EntityId const implicit; blk: lambda<(eid:decs::EntityId const;var cmp:array<decs::ComponentValue> -const):void>*)

argument	argument type
entityid	*decs::EntityId* const implicit
blk	lambda<(eid: *decs::EntityId* const;cmp: *ComponentMap*):void>

Creates deferred action to update entity specified by id.

create_entity (*blk: lambda<(eid:decs::EntityId const;var cmp:array<decs::ComponentValue> -const):void>*)

create_entity returns *decs::EntityId*

argument	argument type
blk	lambda<(eid: *decs::EntityId* const;cmp: *ComponentMap*):void>

Creates deferred action to create entity.

delete_entity (*entityid: EntityId const implicit*)

argument	argument type
entityid	*decs::EntityId* const implicit

Creates deferred action to delete entity specified by id.

48.7 GC and reset

- *restart () : void*
- *before_gc () : void*
- *after_gc () : void*

restart ()

Restarts ECS by erasing all deferred actions and entire state.

before_gc ()

Low level callback to be called before the garbage collection. This is a low-level function typically used by *live*.

after_gc()

Low level callback to be called after the garbage collection. This is a low-level function typically used by *live*.

48.8 Iteration

- *for_each_archetype (erq:decs::EcsRequest -const;blk:block<(arch:decs::Archetype const):void> const) : void*
- *for_eid_archetype (eid:decs::EntityId const implicit;hash:uint64 const;erq:function<decs::EcsRequest> -const;blk:block<(arch:decs::Archetype const;index:int const):void> const) : bool const*
- *for_each_archetype (hash:uint64 const;erq:function<decs::EcsRequest> -const;blk:block<(arch:decs::Archetype const):void> const) : void*
- *for_each_archetype_find (hash:uint64 const;erq:function<decs::EcsRequest> -const;blk:block<(arch:decs::Archetype const):bool> const) : bool const*
- *decs_array (atype:auto(TT) const;src:array<uint8> const;capacity:int const) : auto*
- *get_ro (arch:decs::Archetype const;name:string const;value:auto(TT) const[]) : array<TT[-2] -const -& -#> const*
- *get_ro (arch:decs::Archetype const;name:string const;value:auto(TT) const) : array<TT -const -& -#> const*
- *get_default_ro (arch:decs::Archetype const;name:string const;value:auto(TT) const) : iterator<TT const&>*
- *get_optional (arch:decs::Archetype const;name:string const;value:auto(TT)? const) : iterator<TT -const -& -#?>*

for_each_archetype(*erq: EcsRequest; blk: block<(arch:decs::Archetype const):void> const*)

argument	argument type
erq	*decs::EcsRequest*
blk	block<(arch: *decs::Archetype* const):void> const

Invokes block for each entity of each archetype that can be processed by the request. Request is returned by a specified function.

for_eid_archetype(*eid: EntityId const implicit; hash: ComponentHash; erq: function<decs::EcsRequest>; blk: block<(arch:decs::Archetype const;index:int const):void> const*)

for_eid_archetype returns bool const

argument	argument type
eid	*decs::EntityId* const implicit
hash	*ComponentHash*
erq	function<>
blk	block<(arch: *decs::Archetype* const;index:int const):void> const

Invokes block for the specific entity id, given request. Request is returned by a specified function.

for_each_archetype (*hash: ComponentHash; erq: function<decs::EcsRequest>; blk: block<(arch:decs::Archetype const):void> const*)

argument	argument type
hash	*ComponentHash*
erq	function<>
blk	block<(arch: *decs::Archetype* const):void> const

Invokes block for each entity of each archetype that can be processed by the request. Request is returned by a specified function.

for_each_archetype_find (*hash: ComponentHash; erq: function<decs::EcsRequest>; blk: block<(arch:decs::Archetype const):bool> const*)

for_each_archetype_find returns bool const

argument	argument type
hash	*ComponentHash*
erq	function<>
blk	block<(arch: *decs::Archetype* const):bool> const

Invokes block for each entity of each archetype that can be processed by the request. Request is returned by a specified function. If block returns true, iteration is stopped.

decs_array (*atype: auto(TT) const; src: array<uint8> const; capacity: int const*)

decs_array returns auto

argument	argument type
atype	auto(TT) const
src	array<uint8> const
capacity	int const

Low level function returns temporary array of component given specific type of component.

get_ro (*arch: Archetype const; name: string const; value: auto(TT) const[]*)

get_ro returns array<TT[-2]> const

argument	argument type
arch	*decs::Archetype* const
name	string const
value	auto(TT) const[-1]

Returns const temporary array of component given specific name and type of component for regular components.

get_ro (*arch: Archetype const; name: string const; value: auto(TT) const*)

get_ro returns array<TT> const

argument	argument type
arch	*decs::Archetype* const
name	string const
value	auto(TT) const

Returns const temporary array of component given specific name and type of component for regular components.

get_default_ro (*arch: Archetype const; name: string const; value: auto(TT) const*)

get_default_ro returns iterator<TT const&>

argument	argument type
arch	*decs::Archetype* const
name	string const
value	auto(TT) const

Returns const iterator of component given specific name and type of component. If component is not found - iterator will kepp returning the specified value.

get_optional (*arch: Archetype const; name: string const; value: auto(TT)? const*)

get_optional returns iterator<TT?>

argument	argument type
arch	*decs::Archetype* const
name	string const
value	auto(TT)? const

Returns const iterator of component given specific name and type of component. If component is not found - iterator will kepp returning default value for the component type.

48.9 Request

- *EcsRequestPos (at:rtti::LineInfo const) : decs::EcsRequestPos*
- *verify_request (erq:decs::EcsRequest -const) : tuple<ok:bool;error:string>*
- *compile_request (erq:decs::EcsRequest -const) : void*
- *lookup_request (erq:decs::EcsRequest -const) : int*

EcsRequestPos (*at: LineInfo const*)

EcsRequestPos returns *decs::EcsRequestPos*

argument	argument type
at	*rtti::LineInfo* const

Constructs EcsRequestPos from rtti::LineInfo.

verify_request (*erq: EcsRequest*)

verify_request returns tuple<ok:bool;error:string>

argument	argument type
erq	*decs::EcsRequest*

Verifies ESC request. Returns pair of boolean (true for OK) and error message.

compile_request (*erq: EcsRequest*)

argument	argument type
erq	*decs::EcsRequest*

Compiles ESC request, by creating request hash.

lookup_request (*erq: EcsRequest*)

lookup_request returns int

argument	argument type
erq	*decs::EcsRequest*

Looks up ESC request in the request cache.

CHAPTER

FORTYNINE

BOOST PACKAGE FOR DECS

The DECS_BOOST module implements queries, stages, and templates for the DECS. Under normal circumstances this is the main require module for DECS.

All functions and symbols are in "decs_boost" module, use require to get access to it.

```
require daslib/desc_boost
```

49.1 Type aliases

ItCheck is a variant type

yes	string
no	bool

DECS prefix check.

49.2 Function annotations

REQUIRE

This annotation provides list of required components for entity.

REQUIRE_NOT

This annotation provides list of components, which are required to not be part of the entity.

decs

This macro converts a function into a DECS pass stage query. Possible arguments are *stage*, 'REQUIRE', and *REQUIRE_NOT*. It has all other properties of a *query* (like ability to operate on templates). For example:

```
[decs(stage=update_ai, REQUIRE=ai_turret)]
    def update_ai ( eid:EntityId; var turret:Turret; pos:float3 )
        ...
```

In the example above a query is added to the *update_ai* stage. The query also requires that each entity passed to it has an *ai_turret* property.

49.3 Call macros

query

This macro implmenets 'query` functionality. There are 2 types of queries:

- query(...) - returns a list of entities matching the query
- query(eid) - returns a single entity matching the eid

For example:

```
query() <| $ ( eid:EntityId; pos, vel : float3 )
    print("[{eid}] pos={pos} vel={vel}\n")
```

The query above will print all entities with position and velocity. Here is another example:

```
query(kaboom) <| $ ( var pos:float3&; vel:float3; col:uint=13u )
    pos += vel
```

The query above will add the velocity to the position of an entity with eid kaboom.

Query can have *REQUIRE* and *REQUIRE_NOT* clauses:

```
var average : float3
query <| $ [REQUIRE(tank)] ( pos:float3 )
    average += pos
```

The query above will add *pos* components of all entities, which also have a *tank* component.

Additionally queries can atuomaticall expand components of entities. For example:

```
[decs_template(prefix="particle")]
struct Particle
    pos, vel : float3
...
query <| $ ( var q : Particle )
    q.pos += q.vel                    // this is actually particlepos += particlevel
```

In the example above structure q : Particle does not exist as a variable. Instead it is expanded into accessing individual componentes of the entity. REQURE section of the query is automatically filled with all components of the template. If template prefix is not specified, prefix is taken from the name of the template (would be "**Particle_**"). Specifying empty prefix *[decs_template(prefix)]* will result in no prefix beeing added.

Note: apart from tagging structure as a template, the macro also generates *apply_decs_template* and *remove_decs_template* functions. *apply_decs_template* is used to add template to an entity, and *remove_decs_template* is used to remove all components of the template from the entity:

```
for i in range(3)
    create_entity <| @ ( eid, cmp )
        apply_decs_template(cmp, [[Particle pos=float3(i), vel=float3(i+1)]])
```

find_query

This macro implmenets 'find_query` functionality. It is similar to *query* in most ways, with the main differences being:

- there is no eid-based find query
- the find_query stops once the first match is found

For example:

```
let found = find_query <| $ ( pos,dim:float3; obstacle:Obstacle )
if !obstacle.wall
    return false
let aabb = [[AABB min=pos-dim*0.5, max=pos+dim*0.5 ]]
if is_intersecting(ray, aabb, 0.1, dist)
    return true
```

In the example above the find_query will return *true* once the first intesection is found. Note: if return is missing, or end of find_query block is reached - its assumed that find_query did not find anything, and will return false.

49.4 Structure macros

decs_template

This macro creates a template for the given structure. *apply_decs_template* and *remove_decs_template* functions are generated for the structure type.

CHAPTER

FIFTY

COROUTINES AND ADDITIONAL GENERATOR SUPPORT

The COROUTINES module exposes coroutine infrastructure, as well as additional yielding facilities.

The following example illustrates iterating over the elements of a tree. *each_async_generator* implements straight up iterator, where 'yield_from' helper is used to continue iterating over leafs. *[coroutine]* annotation converts function into coroutine. If need be, return type of the function can specify coroutine yield type:

```
require daslib/coroutines

struct Tree
    data : int
    left, right : Tree?

// yield from example
def each_async_generator(tree : Tree?)
    return <- generator<int>() <|
        if tree.left != null
            yeild_from <| each_async_generator(tree.left)
        yield tree.data
        if tree.right != null
            yeild_from <| each_async_generator(tree.right)
        return false

// coroutine as function
[coroutine]
def each_async(tree : Tree?) : int
    if tree.left != null
        co_await <| each_async(tree.left)
    yield tree.data
    if tree.right != null
        co_await <| each_async(tree.right)
```

All functions and symbols are in "coroutines" module, use require to get access to it.

```
require daslib/coroutines
```

50.1 Type aliases

`Coroutine = iterator<bool>`

Coroutine which does not yield and value.

`Coroutines = array<iterator<bool>>`

Collection of coroutines, which do not yield any value.

50.2 Function annotations

`coroutine`

This macro converts coroutine function into generator, adds return false. Daslang impelmentation of coroutine is generator based. Function is converted into a state machine, which can be resumed and suspended. The function is converted into a generator. Generator yields bool if its a void coroutine, and yields the return type otherwise. If return type is specified coroutine can serve as an advanced form of a generator.

50.3 Call macros

`co_continue`

This macro converts co_continue to yield true. The idea is that coroutine without specified type is underneath a coroutine which yields bool. That way co_continue() does not distract from the fact that it is a generator<bool>.

`co_await`

This macro converts co_await(sub_coroutine) into:

```
for t in subroutine
    yield t
```

The idea is that coroutine or generator can wait for a sub-coroutine to finish.

`yeild_from`

This macro converts yield_from(THAT) expression into:

```
for t in THAT
    yield t
```

The idea is that coroutine or generator can continuesly yield from another sub-coroutine or generator.

50.4 Top level coroutine evaluation

- *cr_run (a:iterator<bool> -const) : void*
- *cr_run_all (a:array<iterator<bool>> -const) : void*

`cr_run` (*a: Coroutine*)

argument	argument type
a	*Coroutine*

This function runs coroutine until it is finished.

cr_run_all (*a: Coroutines*)

argument	argument type
a	*Coroutines*

This function runs all coroutines until they are finished.

CHAPTER FIFTYONE

INTERFACES

The interface module implements [interface] pattern, which allows classes to expose multiple interfaces.

All functions and symbols are in “interfaces” module, use require to get access to it.

```
require daslib/interfaces
```

Lets review the following example:

```
require daslib/interfaces

[interface]
class ITick
    def abstract beforeTick : bool
    def abstract tick ( dt:float ) : void
    def abstract afterTick : void

[interface]
class ILogger
    def abstract log ( message : string ) : void

[implements(ITick),implements(ILogger)]
class Foo
    def Foo
        pass
    def ITick`tick ( dt:float )
        print("tick {dt}\n")
    def ITick`beforeTick
        print("beforeTick\n")
        return true
    def ITick`afterTick
        print("afterTick\n")
    def ILogger`log ( message : string )
        print("log {message}\n")
```

In the example above, we define two interfaces, ITick and ILogger. Then we define a class Foo, which implements both interfaces. The class Foo must implement all methods of both interfaces. The class Foo can implement additional methods, which are not part of the interfaces.

The [implements] attribute is used to specify which interfaces the class implements.

The [interface] attribute is used to define an interface. This macro verifies that the interface does not have any data members, only methods.

Interface methods are automatically bound to specific interfaces, by pattern-matching the method name. For example the method “tick” is bound to the interface ITick, because the method name starts with “ITick`”. The method “log” is bound to the interface ILogger, because the method name starts with “ILogger`”.

Additionally get`ITick and get`ILogger methods are generated for the Foo class. They are used to get the interface object for the given interface. The interface object is used to call the interface methods.

```
var f = new Foo()
f->get`ITick()->beforeTick()
f->get`ITick()->tick(1.0)
f->get`ITick()->afterTick()
f->get`ILogger()->log("hello")
```

51.1 Structure macros

interface

implements 'interface' macro, which verifies if class is an interface (no own variables)

implements

implements 'implements' macro, adds get`{Interface} method as well as interface bindings and implementation.

CHAPTER

FIFTYTWO

EXPORT CONSTRUCTOR

The export_constructor module simplifies creation of Daslang structure and classes from C++ side.

In the following example:

```
[export_constructor]
class Foo {}
```

Function make`Foo is generated with an export flag; it returns new Foo() object.

All functions and symbols are in "export_constructor" module, use require to get access to it.

```
require daslib/export_constructor
```

52.1 Structure macros

export_constructor

implements 'export_constructor' macro, adds function make`{StructureName} which makes a new instance of a class or structure

CHAPTER

FIFTYTHREE

FAKER

The FAKER module implements collection of random data generators for use in testing and otherwise.

All functions and symbols are in "faker" module, use require to get access to it.

```
require daslib/faker
```

53.1 Type aliases

BitRepresentation64 is a variant type

ui2	uint[2]
d	double
i64	int64
u64	uint64

64-bit representation of a float

Faker

Faker fields are

min_year	uint
total_years	uint
rnd	iterator<uint>
max_long_string	uint

Instance of the faker with all the settings inside.

53.2 Constructor

- *Faker (rng:iterator<uint> -const) : faker::Faker*

Faker (*rng: iterator<uint>*)

Faker returns *faker::Faker*

argument	argument type
rng	iterator<uint>

Creates new instance of a *Faker* given a random number generator.

53.3 Random values

- *random_int (faker:faker::Faker -const) : int*
- *random_uint (faker:faker::Faker -const) : uint*
- *random_int8 (faker:faker::Faker -const) : int8*
- *random_uint8 (faker:faker::Faker -const) : uint8*
- *random_int16 (faker:faker::Faker -const) : int16*
- *random_uint16 (faker:faker::Faker -const) : uint16*
- *random_float (faker:faker::Faker -const) : float*
- *random_int2 (faker:faker::Faker -const) : int2*
- *random_range (faker:faker::Faker -const) : range*
- *random_range64 (faker:faker::Faker -const) : range64*
- *random_int3 (faker:faker::Faker -const) : int3*
- *random_int4 (faker:faker::Faker -const) : int4*
- *random_uint2 (faker:faker::Faker -const) : uint2*
- *random_urange (faker:faker::Faker -const) : urange*
- *random_urange64 (faker:faker::Faker -const) : urange64*
- *random_uint3 (faker:faker::Faker -const) : uint3*
- *random_uint4 (faker:faker::Faker -const) : uint4*
- *random_float2 (faker:faker::Faker -const) : float2*
- *random_float3 (faker:faker::Faker -const) : float3*
- *random_float4 (faker:faker::Faker -const) : float4*
- *random_float3x3 (faker:faker::Faker -const) : math::float3x3*
- *random_float3x4 (faker:faker::Faker -const) : math::float3x4*
- *random_float4x4 (faker:faker::Faker -const) : math::float4x4*
- *random_int64 (faker:faker::Faker -const) : int64*

- *random_uint64 (faker:faker::Faker -const) : uint64*
- *random_double (faker:faker::Faker -const) : double*

random_int (*faker: Faker*)

random_int returns int

argument	argument type
faker	*faker::Faker*

Generates random integer.

random_uint (*faker: Faker*)

random_uint returns uint

argument	argument type
faker	*faker::Faker*

Generates random unsigned integer.

random_int8 (*faker: Faker*)

random_int8 returns int8

argument	argument type
faker	*faker::Faker*

Generates random int8.

random_uint8 (*faker: Faker*)

random_uint8 returns uint8

argument	argument type
faker	*faker::Faker*

Generates random uint8.

random_int16 (*faker: Faker*)

random_int16 returns int16

argument	argument type
faker	*faker::Faker*

Generates random int16.

random_uint16 (*faker: Faker*)

random_uint16 returns uint16

argument	argument type
faker	*faker::Faker*

Generates random uint16.

random_float (*faker: Faker*)

random_float returns float

argument	argument type
faker	*faker::Faker*

Generates random float.

random_int2 (*faker: Faker*)

random_int2 returns int2

argument	argument type
faker	*faker::Faker*

Generates random int2.

random_range (*faker: Faker*)

random_range returns range

argument	argument type
faker	*faker::Faker*

Generates random range.

random_range64 (*faker: Faker*)

random_range64 returns range64

argument	argument type
faker	*faker::Faker*

Generates random range64.

random_int3 (*faker: Faker*)

random_int3 returns int3

argument	argument type
faker	*faker::Faker*

Generates random int3.

random_int4 (*faker: Faker*)

random_int4 returns int4

argument	argument type
faker	*faker::Faker*

Generates random int4.

random_uint2 (*faker: Faker*)

random_uint2 returns uint2

argument	argument type
faker	*faker::Faker*

Generates random uint2.

random_urange (*faker: Faker*)

random_urange returns urange

argument	argument type
faker	*faker::Faker*

Generates random urange.

random_urange64 (*faker: Faker*)

random_urange64 returns urange64

argument	argument type
faker	*faker::Faker*

Generates random urange64.

random_uint3 (*faker: Faker*)

random_uint3 returns uint3

argument	argument type
faker	*faker::Faker*

Generates random uint3.

random_uint4 (*faker: Faker*)

random_uint4 returns uint4

argument	argument type
faker	*faker::Faker*

Generates random uint4.

random_float2 (*faker: Faker*)

random_float2 returns float2

argument	argument type
faker	*faker::Faker*

Generates random float2.

random_float3 (*faker: Faker*)

random_float3 returns float3

argument	argument type
faker	*faker::Faker*

Generates random float3.

random_float4 (*faker: Faker*)

random_float4 returns float4

argument	argument type
faker	*faker::Faker*

Generates random float4.

random_float3x3 (*faker: Faker*)

random_float3x3 returns *math::float3x3*

argument	argument type
faker	*faker::Faker*

Generates random float3x3.

random_float3x4 (*faker: Faker*)

random_float3x4 returns *math::float3x4*

argument	argument type
faker	*faker::Faker*

Generates random float3x4.

random_float4x4 (*faker: Faker*)

random_float4x4 returns *math::float4x4*

argument	argument type
faker	*faker::Faker*

Generates random float4x4.

random_int64 (*faker: Faker*)

random_int64 returns int64

argument	argument type
faker	*faker::Faker*

Generates random int64

random_uint64 (*faker: Faker*)

random_uint64 returns uint64

argument	argument type
faker	*faker::Faker*

Generates random uint64

random_double (*faker: Faker*)

random_double returns double

argument	argument type
faker	*faker::Faker*

Generates random double.

53.4 Random strings

- *long_string (faker:faker::Faker -const) : string*
- *any_string (faker:faker::Faker -const) : string*
- *any_file_name (faker:faker::Faker -const) : string*
- *any_set (faker:faker::Faker -const) : uint[8]*
- *any_char (faker:faker::Faker -const) : int*
- *number (faker:faker::Faker -const) : string*
- *positive_int (faker:faker::Faker -const) : string*
- *any_int (faker:faker::Faker -const) : string*
- *any_hex (faker:faker::Faker -const) : string*
- *any_float (faker:faker::Faker -const) : string*
- *any_uint (faker:faker::Faker -const) : string*

long_string (*faker: Faker*)

long_string returns string

argument	argument type
faker	*faker::Faker*

Generates a long string of random characters. The string is anywhere between 0 and faker.max_long_string characters long.

any_string (*faker: Faker*)

any_string returns string

argument	argument type
faker	*faker::Faker*

Generates a string of random characters. The string is anywhere between 0 and regex::re_gen_get_rep_limit() characters long.

any_file_name (*faker: Faker*)

any_file_name returns string

argument	argument type
faker	*faker::Faker*

Generates random file name.

any_set (*faker: Faker*)

any_set returns uint[8]

argument	argument type
faker	*faker::Faker*

Generates random set (uint[8])

any_char (*faker: Faker*)

any_char returns int

argument	argument type
faker	*faker::Faker*

Generates random char. (1 to 255 range)

number (*faker: Faker*)

number returns string

argument	argument type
faker	*faker::Faker*

Generates random number string.

positive_int (*faker: Faker*)

positive_int returns string

argument	argument type
faker	*faker::Faker*

Generates random positive integer string.

any_int (*faker: Faker*)

any_int returns string

argument	argument type
faker	*faker::Faker*

Generates random integer string.

any_hex (*faker: Faker*)

any_hex returns string

argument	argument type
faker	*faker::Faker*

Generates random integer hex string.

any_float (*faker: Faker*)

any_float returns string

argument	argument type
faker	*faker::Faker*

Generates random float string.

any_uint (*faker: Faker*)

any_uint returns string

argument	argument type
faker	*faker::Faker*

Generates random unsigned integer string.

53.5 Date and time

- *month (faker:faker::Faker -const) : string*
- *day (faker:faker::Faker -const) : string*
- *is_leap_year (year:uint const) : bool*
- *week_day (year:uint const;month:uint const;day:uint const) : int*
- *week_day (year:int const;month:int const;day:int const) : int*
- *date (faker:faker::Faker -const) : string*

month (*faker: Faker*)

month returns string

argument	argument type
faker	*faker::Faker*

Generates random month string.

day (*faker: Faker*)

day returns string

argument	argument type
faker	*faker::Faker*

Generates random day string.

is_leap_year (*year: uint const*)

is_leap_year returns bool

argument	argument type
year	uint const

Returns true if year is leap year.

week_day (*year: uint const; month: uint const; day: uint const*)

week_day returns int

argument	argument type
year	uint const
month	uint const
day	uint const

Returns week day for given date.

week_day (*year: int const; month: int const; day: int const*)

week_day returns int

argument	argument type
year	int const
month	int const
day	int const

Returns week day for given date.

date (*faker: Faker*)

date returns string

argument	argument type
faker	*faker::Faker*

Generates random date string.

CHAPTER

FIFTYFOUR

FUZZER

The FUZZER module implements facilities for the fuzz testing.

The idea behind the fuzz testing is to feed random data to the testing function and see if it crashes. *panic* is considered a valid behavior, and in fact ignored. Fuzz tests work really well in combination with the sanitizers (asan, ubsan, etc).

All functions and symbols are in "fuzzer" module, use require to get access to it.

```
require daslib/fuzzer
```

54.1 Fuzzer tests

- *fuzz (blk:block<> const) : void*
- *fuzz (fuzz_count:int const;blk:block<> const) : void*
- *fuzz_debug (blk:block<> const) : void*
- *fuzz_debug (fuzz_count:int const;blk:block<> const) : void*
- *fuzz_numeric_and_vector_op1 (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*
- *fuzz_numeric_and_vector_signed_op1 (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*
- *fuzz_numeric_op1 (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*
- *fuzz_numeric_and_storage_op1 (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*
- *fuzz_all_ints_op1 (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*
- *fuzz_all_unsigned_ints_op1 (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*
- *fuzz_float_double_or_float_vec_op1 (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*
- *fuzz_float_or_float_vec_op1 (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*
- *fuzz_numeric_and_vector_op2 (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*
- *fuzz_numeric_and_vector_op2_no_unint_vec (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*
- *fuzz_numeric_op2 (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*
- *fuzz_compareable_op2 (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*
- *fuzz_eq_neq_op2 (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*
- *fuzz_numeric_vec_scal_op2 (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*

- *fuzz_numeric_scal_vec_op2 (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*
- *fuzz_int_vector_op2 (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*
- *fuzz_shift_op2 (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*
- *fuzz_rotate_op2 (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*
- *fuzz_numeric_op3 (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*
- *fuzz_vec_op3 (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*
- *fuzz_vec_mad_op3 (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*
- *fuzz_float_double_or_float_vec_op3 (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*
- *fuzz_numeric_op4 (t:testing::T? const;fake:faker::Faker -const;funcname:string const) : void*

fuzz (*blk: block<> const*)

argument	argument type
blk	block<> const

run block however many times ignore panic, so that we can see that runtime crashes

fuzz (*fuzz_count: int const; blk: block<> const*)

argument	argument type
fuzz_count	int const
blk	block<> const

run block however many times ignore panic, so that we can see that runtime crashes

fuzz_debug (*blk: block<> const*)

argument	argument type
blk	block<> const

run block however many times do not ignore panic, so that we can see where the runtime fails this is here so that *fuzz* can be easily replaced with *fuzz_debug* for the purpose of debugging

fuzz_debug (*fuzz_count: int const; blk: block<> const*)

argument	argument type
fuzz_count	int const
blk	block<> const

run block however many times do not ignore panic, so that we can see where the runtime fails this is here so that *fuzz* can be easily replaced with *fuzz_debug* for the purpose of debugging

fuzz_numeric_and_vector_op1 (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes single numeric or vector argument. arguments are: int, uint, float, double, string, int2, int3, int4, uint2, uint3, uint4, float2, float3, float4

fuzz_numeric_and_vector_signed_op1 (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes single numeric or vector argument. arguments are: int, uint, float, double, string, int2, int3, int4, uint2, uint3, uint4, float2, float3, float4

fuzz_numeric_op1 (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes single numeric or vector argument. arguments are: int, uint, float, double

fuzz_numeric_and_storage_op1 (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes single numeric or vector argument. arguments are: int, uint, int8, uint8, int16, uint16, int64, uint64, float, double

fuzz_all_ints_op1 (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes single numeric or vector argument. arguments are: int, uint, int64, uint64

fuzz_all_unsigned_ints_op1 (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes single numeric or vector argument. arguments are: uint, uint64

fuzz_float_double_or_float_vec_op1 (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes single numeric or vector argument. arguments are: float, double, float2, float3, float4

fuzz_float_or_float_vec_op1 (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes single numeric or vector argument. arguments are: float, float2, float3, float4

fuzz_numeric_and_vector_op2 (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes two numeric or vector arguments. arguments are: int, uint, float, double, int2, int3, int4, uint2, uint3, uint4, float2, float3, float4

fuzz_numeric_and_vector_op2_no_unint_vec (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes two numeric or vector arguments. arguments are: int, uint, float, double, int2, int3, int4, float2, float3, float4

fuzz_numeric_op2 (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes two numeric or vector arguments. arguments are: int, uint, float, double

fuzz_compareable_op2 (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes two numeric or vector arguments. arguments are: int, uint, float, double, int64, uint64, string

fuzz_eq_neq_op2 (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes two numeric or vector arguments. arguments are: int, uint, int64, uint64, float, double, string, int2, int3, int4, uint2, uint3, uint4, float2, float3, float4

fuzz_numeric_vec_scal_op2 (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes vector and matching scalar on the right arguments pairs are: int2,int; int3,int; uint2,uint; uint3,uint; uint4,uint; int4,int; float2,float; float3,float; float4,float

fuzz_numeric_scal_vec_op2 (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes vector and matching scalar on the left arguments pairs are: int2,int; int3,int; uint2,uint; uint3,uint; uint4,uint; int4,int; float2,float; float3,float; float4,float

fuzz_int_vector_op2 (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes two numeric or vector arguments. arguments are: int, uint, int2, int3, int4, uint2, uint3, uint4

fuzz_shift_op2 (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes numeric or vector argument, with matching shift type on the right. arguments are: int, uint, int2, int3, int4, uint2, uint3, uint4

fuzz_rotate_op2 (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes numeric or vector argument, with matching rotate type on the right. arguments are: int, uint

fuzz_numeric_op3 (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes three numeric or vector arguments. arguments are: int, uint, float, double

fuzz_vec_op3 (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes three numeric or vector arguments. arguments are: float2, float3, float4, int2, int3, int4, uint2, uint3, uint4

fuzz_vec_mad_op3 (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes three numeric or vector arguments. arguments are: float2, float3, float4, int2, int3, int4, uint2, uint3, uint4 second argument is float, int, uint accordingly

fuzz_float_double_or_float_vec_op3 (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes three numeric or vector arguments. arguments are: float, double, float2, float3, float4

fuzz_numeric_op4 (*t: testing::T? const; fake: Faker; funcname: string const*)

argument	argument type
t	testing::T ? const
fake	*faker::Faker*
funcname	string const

fuzzes generic function that takes four numeric or vector arguments. arguments are: int, uint, float, double

CHAPTER FIFTYFIVE

PATTERN MATCHING

The MATCH module implements pattern matching in Daslang. (See also the pattern-matching section.)

All functions and symbols are in “match” module, use require to get access to it.

```
require daslib/match
```

55.1 Call macros

match

Implements *match* macro.

static_match

Implements *static_match* macro.

multi_match

Implements *multi_match* macro.

static_multi_match

Implements *static_multi_match* macro.

55.2 Structure macros

match_as_is

Implements *match_as_is* annotation. This annotation is used to mark that structure can be matched with different type via is and as machinery.

match_copy

Implements *match_copy* annotation. This annotation is used to mark that structure can be matched with different type via match_copy machinery.