DataSpecification Documentation

Jan 29, 2020

Contents

1 Functional Requirements **3**

2 Use Cases **5**

3 Contents **7**

3.1 data_specification package 7
3.1.1 Subpackages 7
3.1.1.1 data_specification.enums package 7
3.1.2 Submodules 14
3.1.3 data_specification.abstract_executor_functions module 14
3.1.4 data_specification.constants module 15
3.1.5 data_specification.data_specification_executor module 15
3.1.6 data_specification.data_specification_executor_functions module 17
3.1.7 data_specification.data_specification_generator module 19
3.1.8 data_specification.exceptions module 44
3.1.9 data_specification.memory_region module 49
3.1.10 data_specification.memory_region_collection module 50
3.1.11 data_specification.utility_calls module 50
3.1.12 Module contents 51
3.1.12.1 Functional Requirements 52
3.1.12.2 Use Cases 52

4 Indices and tables **83**

Python Module Index **85**

Index **87**

These pages document the python code for the DataSpecification module which is part of the SpiNNaker Project.

This code depends on SpiNNUtils, SpiNNMachine and SpiNNStorageHandlers (Combined_documentation).

Used to generate memory images for a SpiNNaker CPU core from a set of instructions.

The main part of this package is the *data_specification.DataSpecificationGenerator* class. This is used to generate a "Data Specification", which can then be executed to produce a memory image. This package also handles this function if required, through the *data_specification.DataSpecificationExecutor* class.

CHAPTER 1

Functional Requirements

Creation of a Data Specification Language file which can be executed to produce a memory image.

- Any errors that can be checked during the creation of the specification should throw an exception.
- It will be impossible to detect all errors at creation time.
- There should be no assumption of where the data specification will be stored, although a default provision of a way to write the specification to a file is acceptable.

Execution of a Data Specification Language file, producing a memory image.

- This should detect any errors during execution and report them, halting the execution.
- There should be no assumption of where the data specification is read from, although a default provision of a way to read the specification from a file is acceptable.

CHAPTER 2

Use Cases

There are a number of use-cases of this library:

- *DataSpecificationGenerator* is used to create a compressed memory image which can be expanded later, to reduce the amount of data that needs to be transferred over a slow connection.
- *DataSpecificationExecutor* is used to execute a previously generated specification at the receiving end of a slow connection.

CHAPTER 3

Contents

3.1 data_specification package

3.1.1 Subpackages

3.1.1.1 data_specification.enums package

Submodules

data_specification.enums.arithemetic_operation module

class data_specification.enums.arithemetic_operation.**ArithmeticOperation**(*value*, *operator*, *doc=''*)

Bases: enum.Enum

Arithmetic Operations

ADD = 0

MULTIPLY = 2

SUBTRACT = 1

data_specification.enums.commands module

class data_specification.enums.commands.**Commands**(*value*, *exec_function*, *doc=''*)

Bases: enum.Enum

Set of opcodes for the spec executor

ALIGN_WR_PTR = 101

ARITH_OP = 103

BLOCK_COPY = 69

BREAK = 0

BREAK_LOOP = 82

CONSTRUCT = 64

COPY_PARAM = 113

COPY_STRUCT = 112

DECLARE_RANDOM_DIST = 6

DECLARE_RNG = 5

ELSE = 86

END_CONSTRUCTOR = 37

END_IF = 87

END_LOOP = 83

END_SPEC = 255

END_STRUCT = 18

FREE = 3

GET_RANDOM_NUMBER = 7

GET_WR_PTR = 99

IF = 85

LOGIC_OP = 104

LOOP = 81

MV = 96

NOP = 1

PRINT_STRUCT = 130

PRINT_TXT = 129

PRINT_VAL = 128

READ = 65

READ_PARAM = 115

REFORMAT = 106

RESERVE = 2

SET_WR_PTR = 100

START_CONSTRUCTOR = 32

START_STRUCT = 16

STRUCT_ELEM = 17

SWITCH_FOCUS = 80

WRITE = 66

WRITE_ARRAY = 67

WRITE_PARAM = 114

WRITE_PARAM_COMPONENT = 116

WRITE_STRUCT = 68

data_specification.enums.condition module

class data_specification.enums.condition.**Condition**(*value*, *operator*, *doc=''*)
Bases: enum.Enum

Comparison Operations

EQUAL = 0

GREATER_THAN = 5

GREATER_THAN_OR_EQUAL = 4

LESS_THAN = 3

LESS_THAN_OR_EQUAL = 2

NOT_EQUAL = 1

data_specification.enums.data_type module

class data_specification.enums.data_type.**DataType**
Bases: enum.Enum

Supported data types The first value is an identifier for the enum class; The second value is the size in bytes of the type; The third value is the minimum possible value for the type; The fourth value is the maximum possible value for the type; The fifth value is the scale of the input value to convert it in integer; The sixth value is the pattern to use following the struct package encodings to convert the data in binary format; The seventh value is whether to apply the scaling when converting to SpiNNaker's binary format. The eighth value is the corresponding numpy type (or None to inhibit direct conversion via numpy, scaled conversion still supported); The ninth value is the text description of the type.

FLOAT_32 = 14

FLOAT_64 = 15

INT16 = 5

INT32 = 6

INT64 = 7

INT8 = 4

S015 = 21

S031 = 22

S063 = 23

S07 = 20

S1615 = 12

S3231 = 13

S87 = 11

U016 = 17

U032 = 18

U064 = 19

U08 = 16

U1616 = 9

U3232 = 10

U88 = 8

UINT16 = 1

UINT32 = 2

UINT64 = 3

UINT8 = 0

encode(*value*)
Encode the Python value for SpiNNaker according to this type.

encode_as_int(*value*)
Returns the value as an integer, according to this type.

max

min

numpy_typename

scale

size

struct_encoding

data_specification.enums.logic_operation module

class data_specification.enums.logic_operation.**LogicOperation**(*value*, *operator*, *doc=''*)

Bases: enum.Enum

Logic Operations

AND = 3

LEFT_SHIFT = 0

NOT = 5

OR = 2

RIGHT_SHIFT = 1

XOR = 4

data_specification.enums.random_number_generator module

class data_specification.enums.random_number_generator.**RandomNumberGenerator**(*value*, *doc=''*)

Bases: enum.Enum

Random number generator types

MERSENNE_TWISTER = 0

Module contents

class data_specification.enums.**ArithmeticOperation**(*value*, *operator*, *doc=''*)

Bases: enum.Enum

Arithmetic Operations

ADD = 0

MULTIPLY = 2

SUBTRACT = 1

class data_specification.enums.**Commands**(*value*, *exec_function*, *doc=''*)

Bases: enum.Enum

Set of opcodes for the spec executor

ALIGN_WR_PTR = 101

ARITH_OP = 103

BLOCK_COPY = 69

BREAK = 0

BREAK_LOOP = 82

CONSTRUCT = 64

COPY_PARAM = 113

COPY_STRUCT = 112

DECLARE_RANDOM_DIST = 6

DECLARE_RNG = 5

ELSE = 86

END_CONSTRUCTOR = 37

END_IF = 87

END_LOOP = 83

END_SPEC = 255

END_STRUCT = 18

FREE = 3

GET_RANDOM_NUMBER = 7

GET_WR_PTR = 99

IF = 85

LOGIC_OP = 104

LOOP = 81

MV = 96

NOP = 1

PRINT_STRUCT = 130

PRINT_TXT = 129

PRINT_VAL = 128

READ = 65

READ_PARAM = 115

REFORMAT = 106

RESERVE = 2

SET_WR_PTR = 100

START_CONSTRUCTOR = 32

START_STRUCT = 16

STRUCT_ELEM = 17

SWITCH_FOCUS = 80

WRITE = 66

WRITE_ARRAY = 67

WRITE_PARAM = 114

WRITE_PARAM_COMPONENT = 116

WRITE_STRUCT = 68

class data_specification.enums.**Condition**(*value*, *operator*, *doc=''*)

Bases: enum.Enum

Comparison Operations

EQUAL = 0

GREATER_THAN = 5

GREATER_THAN_OR_EQUAL = 4

LESS_THAN = 3

LESS_THAN_OR_EQUAL = 2

NOT_EQUAL = 1

class data_specification.enums.**DataType**

Bases: enum.Enum

Supported data types The first value is an identifier for the enum class; The second value is the size in bytes of the type; The third value is the minimum possible value for the type; The fourth value is the maximum possible value for the type; The fifth value is the scale of the input value to convert it in integer; The sixth value is the pattern to use following the struct package encodings to convert the data in binary format; The seventh value is whether to apply the scaling when converting to SpiNNaker's binary format. The eighth value is the corresponding numpy type (or None to inhibit direct conversion via numpy, scaled conversion still supported); The ninth value is the text description of the type.

FLOAT_32 = 14

FLOAT_64 = 15

INT16 = 5

INT32 = 6

INT64 = 7

INT8 = 4

S015 = 21

S031 = 22

S063 = 23

S07 = 20

S1615 = 12

S3231 = 13

S87 = 11

U016 = 17

U032 = 18

U064 = 19

U08 = 16

U1616 = 9

U3232 = 10

U88 = 8

UINT16 = 1

UINT32 = 2

UINT64 = 3

UINT8 = 0

encode(*value*)
Encode the Python value for SpiNNaker according to this type.

encode_as_int(*value*)
Returns the value as an integer, according to this type.

max

min

numpy_typename

scale

size

struct_encoding

class data_specification.enums.**LogicOperation**(*value*, *operator*, *doc=''*)
Bases: enum.Enum

Logic Operations

AND = 3

LEFT_SHIFT = 0

NOT = 5

OR = 2

RIGHT_SHIFT = 1

XOR = 4

class data_specification.enums.**RandomNumberGenerator**(*value*, *doc=''*)
Bases: enum.Enum

Random number generator types

MERSENNE_TWISTER = 0

3.1.2 Submodules

3.1.3 data_specification.abstract_executor_functions module

class data_specification.abstract_executor_functions.**AbstractExecutorFunctions**
Bases: object

This class defines a function related to each of the commands of the data specification file. Subclasses need to provide implementations that work for the operations they wish to support.

execute_align_wr_ptr(*cmd*)

execute_arith_op(*cmd*)

execute_block_copy(*cmd*)

execute_break(*cmd*)

execute_break_loop(*cmd*)

execute_construct(*cmd*)

execute_copy_param(*cmd*)

execute_copy_struct(*cmd*)

execute_declare_rng(*cmd*)

execute_else(*cmd*)

execute_end_constructor(*cmd*)

execute_end_if(*cmd*)

execute_end_loop(*cmd*)

execute_end_spec(*cmd*)

execute_end_struct(*cmd*)

execute_free(*cmd*)

execute_get_random_rumber(*cmd*)

execute_get_wr_ptr(*cmd*)

execute_if(*cmd*)

execute_logic_op(*cmd*)

execute_loop(*cmd*)

execute_mv(*cmd*)

execute_nop(*cmd*)
This command executes no operation.

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

execute_print_struct(*cmd*)

execute_print_txt(*cmd*)

execute_print_val(*cmd*)

execute_random_dist(*cmd*)

execute_read(*cmd*)

execute_read_param(*cmd*)

execute_reformat(*cmd*)

execute_reserve(*cmd*)

execute_reset_wr_ptr(*cmd*)

execute_set_wr_ptr(*cmd*)

execute_start_constructor(*cmd*)

execute_start_struct(*cmd*)

execute_struct_elem(*cmd*)

execute_switch_focus(*cmd*)

execute_write(*cmd*)

execute_write_array(*cmd*)

execute_write_param(*cmd*)

execute_write_param_component(*cmd*)

execute_write_struct(*cmd*)

3.1.4 data_specification.constants module

Constants used by the Data Structure Generator (DSG) and the Specification Executor.

3.1.5 data_specification.data_specification_executor module

class data_specification.data_specification_executor.**DataSpecificationExecutor**(*spec_reader*, *memory_space*)

Bases: object

Used to execute a SpiNNaker data specification language file to produce a memory image

Parameters

- **spec_reader** (AbstractDataReader) – The object to read the specification language file from
- **memory_space** (*int*) – memory available on the destination architecture

Raises

- **spinn_storage_handlers.exceptions.DataReadException** – If a read from external storage fails
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails

dsef

execute()
Executes the specification

Returns Nothing

Raises

- **spinn_storage_handlers.exceptions.DataReadException** – If a read from external storage fails
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.DataSpecificationException*** – If there is an error when executing the specification
- **data_specification.exceptions.TablePointerOutOfMemory** – If the table pointer generated as data header exceeds the size of the available memory

get_constructed_data_size()
Return the size of the data that will be written to memory

Returns size of the data that will be written to memory

Return type int

get_header()
Get the header of the data as a numpy array

get_pointer_table(*start_address*)
Get the pointer table as a numpy array

Parameters **start_address** – The base address of the data to be written

Return type numpy.array

get_region(*region_id*)
Get a region with a given ID

Parameters **region_id** (*int*) – The ID of the region to get

Returns The region, or None if the region was not allocated

Return type MemoryRegion

spec_reader

3.1.6 data_specification.data_specification_executor_functions module

class data_specification.data_specification_executor_functions.**DataSpecificationExecutorFu**

Bases: *data_specification.abstract_executor_functions.AbstractExecutorFunctions*

This class includes the function related to each of the commands of the data specification file.

Parameters

- **spec_reader** (AbstractDataReader) – The object to read the specification language file from
- **memory_space** (*int*) – Memory space available for the data to be generated

cmd_size

current_region

data_len

dest_reg

execute_break(*cmd*)

This command raises an exception to stop the execution of the data spec executor (DSE)

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

Raises *data_specification.exceptions.ExecuteBreakInstruction* – Raises the exception to break the execution of the DSE

execute_end_spec(*cmd*)

Return the value which terminates the data spec executor

Parameters **cmd** (*int*) – the command which triggered the function call

Returns END_SPEC_EXECUTOR

Return type int

Raises *data_specification.exceptions.DataSpecificationSyntaxError* – If command END_SPEC != -1

execute_mv(*cmd*)

This command moves an immediate value to a register or copies the value of a register to another register

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

Raises *data_specification.exceptions.DataSpecificationSyntaxError* – If the destination register is not correctly specified; the destination must be a register and the appropriate bit needs to be set in the specification

execute_reserve(*cmd*)

This command reserves a region and assigns some memory space to it

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

Raises

- ***data_specification.exceptions.DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***data_specification.exceptions.ParameterOutOfBoundsException*** – If the requested size of the region is beyond the available memory space

execute_set_wr_ptr(*cmd*)

execute_switch_focus(*cmd*)

This command switches the focus to the desired, already allocated, memory region

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

Raises ***data_specification.exceptions.RegionUnfilledException*** – If the focus is being switched to a region of memory which has been declared to be kept unfilled

execute_write(*cmd*)

This command writes the given value in the specified region a number of times as identified by either a value in the command or a register value

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

Raises ***data_specification.exceptions.DataSpecificationSyntaxError*** – If there is an error in the command syntax

execute_write_array(*cmd*)

This command writes an array of values in the specified region

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

mem_regions

memory_space

opcode

registers

space_allocated

spec_reader

src1_reg

src2_reg

3.1.7 data_specification.data_specification_generator module

class data_specification.data_specification_generator.**DataSpecificationGenerator**(*spec_writer*, *report_writer=N*

Bases: object

Used to generate a SpiNNaker data specification language file that can be executed to produce a memory image

Parameters

- **spec_writer** (AbstractDataWriter) – The object to write the specification to
- **report_writer** (AbstractDataWriter) – Determines if a text version of the specification is to be written and, if so, where. No report is written if this is None.

Raises **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails

MAGIC_NUMBER = 2903706326

VERSION = 1

align_write_pointer(*log_block_size*, *log_block_size_is_register=False*, *return_register_id=None*)

Insert command to align the write pointer against a power-of-2 block size in bytes. Zeros are inserted in the intervening space

Parameters

- **log_block_size** (*int*) –
  - If log_block_size_is_register is False, log to base 2 of the block size (e.g. The write pointer will be moved so that it is a multiple of 2^(log_block_size)), between 0 and 32
  - If log_block_size_is_register is True, the ID of the register containing log to the base 2 of the block size, between 0 and 15
- **log_block_size_is_register** (*bool*) – Indicates if log_block_size is a register ID
- **return_register_id** (*int*) – The ID of a register where the write pointer will be written to once it has been updated, between 0 and 15or None if no such writing is to be done

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If log_block_size_is_register is False, and log_block_size is not within the allowed range
  - If log_block_size_is_register is True and log_block_size is not a valid register ID

- *data_specification.exceptions.RegionOutOfBoundsException* – If the move of the pointer would put it outside of the current region
- *data_specification.exceptions.NoRegionSelectedException* – If no region has been selected

break_loop()

Insert command to break out of a loop before it has completed

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- *data_specification.exceptions.InvalidCommandException* – If there is no loop in operation at this point

call_arithmetic_operation(*register_id*, *operand_1*, *operation*, *operand_2*, *signed*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform an arithmetic operation on two signed or unsigned values and store the result in a register

Parameters

- **register_id** (*int*) – The ID of the register to store the result in
- **operand_1** (*int*) –
  - If operand_1_is_register is True, the ID of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operation** (ArithmeticOperation) – The operation to perform
- **operand_2** (*int*) –
  - If operand_2_is_register is True, the ID of a register where the second operand can be found, between 0 and 15
  - If operand_2_is_register is False, a 32-bit value
- **signed** (*bool*) – Indicates if the values should be considered signed
- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register ID
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register ID

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- *data_specification.exceptions.ParameterOutOfBoundsException* –
  - If operand_1_is_register is True and operand_1 is not a valid register ID

- If operand_2_is_register is True and operand_2 is not a valid register ID
- ***data_specification.exceptions.UnknownTypeException*** – If operation is not a known operation

call_function(*function_id*, *structure_ids*)
Insert command to call a function

Parameters

- **function_id** (*int*) – The ID of a previously defined function, between 0 and 31
- **structure_ids** (*list of int*) – A list of structure_ids that will be passed into the function, each between 0 and 15

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If the function ID is not valid
  - If any of the structure IDs are not valid
- ***data_specification.exceptions.NotAllocatedException*** –
  - If a function has not been defined with the given ID
  - If no structure has been defined with one of the IDs in structure_ids
- ***data_specification.exceptions.WrongParameterNumberException*** – If a function is called with a wrong number of parameters
- ***data_specification.exceptions.DuplicateParameterException*** – If a function is called with duplicate parameters

call_random_distribution(*distribution_id*, *register_id*)
Insert command to get the next random number from a random distribution, placing the result in a register to be used in a future call

Parameters

- **distribution_id** (*int*) – The ID of the random distribution to call between 0 and 63
- **register_id** (*int*) – The ID of the register to store the result in between 0 and 15

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised

- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.NotAllocatedException*** – If the random distribution ID was not previously declared
- ***data_specification.exceptions.ParameterOutOfBoundsException*** – If the distribution_id or register_id specified was out of range

comment(*comment*)

Write a comment to the text version of the specification. Note that this is ignored by the binary file

Parameters **comment** (*str*) – The comment to write

Returns Nothing is returned

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails

conditionals

copy_structure(*source_structure_id*, *destination_structure_id*, *source_id_is_register=False*, *destination_id_is_register=False*)

Insert command to copy a structure, possibly overwriting another structure

Parameters

- **source_structure_id** (*int*) –
  - If source_id_is_register is True, the ID of the register holding the source structure ID, between 0 and 15
  - Otherwise, the ID of the source structure, between 0 and 15
- **destination_structure_id** (*int*) –
  - If destination_id_is_register is True, the ID of the register holding the destination structure ID, between 0 and 15
  - If destination_id_is_register is False, the ID of the destination structure, between 0 and 15
- **source_id_is_register** (*bool*) – Indicates if source_structure_id is a register ID
- **destination_id_is_register** (*bool*) – Indicates if destination_structure_id is a register ID

Returns The ID of the copied structure, between 0 and 15

Return type int

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –

- If source_id_is_register is True and source_structure_id is not a valid register ID
- If destination_id_is_register is True and destination_structure_id is not a valid register ID
- If source_id_is_register is False and source_structure_id is not a valid structure ID
- If destination_id_is_register is False and destination_structure_id is not a valid structure ID

- ***data_specification.exceptions.NotAllocatedException*** –
  - If no structure with ID source_structure_id has been allocated

copy_structure_parameter(*source_structure_id*, *source_parameter_index*, *destination_id*, *destination_parameter_index=None*, *destination_is_register=False*)

Insert command to copy the value of a parameter from one structure to another.

Parameters

- **source_structure_id** (*int*) – The ID of the source structure, between 0 and 15
- **source_parameter_index** (*int*) – The index of the parameter in the source structure
- **destination_id** (*int*) – The ID of the destination structure, or the ID of the destination register, between 0 and 15
- **destination_parameter_index** (*int*) – The index of the parameter in the destination structure. Ignored when writing to a register.
- **destination_is_register** (*bool*) – Indicates whether the destination is a structure or a register.

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If source_structure_id is not a valid structure ID
  - If destination_id is not a valid structure ID
  - If source_parameter_index is not a valid parameter index in the source structure
  - If destination_parameter_index is not a valid parameter index in the destination structure
- ***data_specification.exceptions.NotAllocatedException*** –
  - If no structure with ID destination_id has been allocated
  - If no structure with ID source_structure_id has been allocated

create_cmd(*data*, *data_type=<DataType.UINT32: 2>*)

Creates command to write a value to the current write pointer, causing the write pointer to move on by the number of bytes required to represent the data type. The data is passed as a parameter to this function.

Note: This does not actually insert the WRITE command in the spec; that is done by *write_cmd()*.

Parameters

- **data** (*int or float*) – the data to write.
- **data_type** (DataType) – the type to convert data to

Returns cmd_word_list; list of binary words to be added to the binary data specification file, and cmd_string; string describing the command to be added to the report for the data specification file

Return type (bytearray, str)

Raises

- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If repeats_register is None, and repeats is out of range
  - If repeats_register is not a valid register ID
  - If data_type is an integer type, and data has a fractional part
  - If data would overflow the data type
- ***data_specification.exceptions.UnknownTypeException*** – If the data type is not known
- ***data_specification.exceptions.InvalidSizeException*** – If the data size is invalid

current_region

declare_random_number_generator(*rng_id*, *rng_type*, *seed*)

Insert command to declare a random number generator

Parameters

- **rng_id** (*int*) – The ID of the random number generator
- **rng_type** (RandomNumberGenerator) – The type of the random number generator
- **seed** (*int*) – The seed of the random number generator >= 0

Returns The ID of the created random number generator, between 0 and 15

Return type int

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.UnknownTypeException*** – If the rng_type is not one of the allowed values
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –

- If the seed is too big or too small
- If the rng_id is not in the allowed range

- ***data_specification.exceptions.RNGInUseException*** – If the random number generator with the given ID has already been defined

declare_uniform_random_distribution(*distribution_id*, *structure_id*, *rng_id*, *min_value*, *max_value*)

Insert commands to declare a uniform random distribution

Parameters

- **distribution_id** – ID of the distribution being set up
- **distribution_id** – int
- **structure_id** (*int*) – ID of an empty structure slot to fill with the uniform random distribution data
- **rng_id** (*int*) – The ID of the random number generator, between 0 and 15
- **min_value** (*float*) – The minimum value that should be returned from the distribution between -32768.0 and max_value
- **max_value** (*float*) – The maximum value that should be returned from the distribution between min_value and 32767.9999847

Returns The ID of the created uniform random distribution to be used in future calls of the distribution between 0 and 63

Return type int

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.NoMoreException*** – If there is no more space for a new random distribution
- ***data_specification.exceptions.NotAllocatedException*** – If the requested rng_id has not been allocated
- ***data_specification.exceptions.ParameterOutOfBoundsException*** – If rng_id, structure_id, min_value or max_value is out of range
- ***data_specification.exceptions.StructureInUseException*** – If structure structure_id is already defined

define_break()

Insert command to stop execution with an exception (for debugging)

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails

define_structure(*structure_id*, *parameters*)
Insert commands to define a data structure

Parameters

- **structure_id** (*int*) – the ID of the structure to create, between 0 and 15
- **parameters** (list of (str, DataType, float)) – A list of between 1 and 255 tuples of (label, data_type, value) where: * label is the label of the element (for debugging) * data_type is the data type of the element * value is the value of the element, or None if no value is to be assigned

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.NoMoreException*** – If there are no more spaces for new structures
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If there are an incorrect number of parameters
  - If the size of one of the tuples is incorrect
  - If one of the values to be assigned has an integer data_type but has a fractional part
  - If one of the values to be assigned would overflow its data_type
- ***data_specification.exceptions.UnknownTypeException*** – If one of the data types in the structure is unknown

else_conditional()
Insert command for the else of an if...then...else construct. If the condition of the conditional evaluates to false, the statements up between the conditional and the insertion of this "else" are skipped, and the statements from this point until the end of the conditional are executed.

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.InvalidCommandException*** – If there is no conditional in operation at this point

end_conditional()
Insert command to mark the end of an if...then...else construct

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.InvalidCommandException*** – If there is no conditional in operation at this point

end_function()
Insert command to mark the end of a function definition

Returns Nothing is returned

Return type None

Raises ***data_specification.exceptions.InvalidCommandException*** – If there is no function being defined at this point

end_loop()
Insert command to indicate that this is the end of the loop. Commands between the start of the loop and this command will be repeated.

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.InvalidCommandException*** – If there is no loop in operation at this point

end_specification(*close_writer=True*)
Insert a command to indicate that the specification has finished and finish writing

Parameters **close_writer** (*bool*) – Indicates whether to close the underlying writer(s)

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails

free_memory_region(*region*)
Insert command to free a previously reserved memory region

Parameters **region** (*int*) – The number of the region to free, from 0 to 15

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.NotAllocatedException*** – If the region was not reserved
- ***data_specification.exceptions.ParameterOutOfBoundsException*** – If the region requested was out of the allowed range

function_slots

get_structure_value(*destination_id*, *structure_id*, *parameter_index*, *parameter_index_is_register=False*)

Insert command to get a value from a structure. The value is copied in a register.

Parameters

- **destination_id** (*int*) – The ID of the destination register
- **structure_id** (*int*) – The ID of the source structure
- **parameter_index** (*int*) – The ID of the parameter/element to copy
- **parameter_index_is_register** (*bool*) – True if the index of the structure is contained in a register

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If structure_id is not in the allowed range
  - If parameter_index is larger than the number of parameters declared in the original structure
  - If destination_id is not the ID of a valid register
  - If parameter_index_is_register is True and parameter_index is not a valid register ID
- ***data_specification.exceptions.NotAllocatedException*** – If the structure requested has not been declared

instruction_counter

logical_and(*register_id*, *operand_1*, *operand_2*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform a logical AND operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The ID of the register to store the result in

- **operand_1** (*int*) –
  - If operand_1_is_register is True, the ID of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operand_2** (*int*) –
  - If operand_2_is_register is True, the ID of a register where the second operand can be found. between 0 and 15
  - If operand_2_is_register is False, a 32-bit value
- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register ID
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register ID

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If operand_1_is_register is True and operand_1 is not a valid register ID
  - If operand_2_is_register is True and operand_2 is not a valid register ID

logical_left_shift(*register_id*, *operand_1*, *operand_2*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform a logical left shift operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The ID of the register to store the result in
- **operand_1** (*int*) –
  - If operand_1_is_register is True, the ID of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operand_2** (*int*) –
  - If operand_2_is_register is True, the ID of a register where the second operand can be found. between 0 and 15
  - If operand_2_is_register is False, a 32-bit value
- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register ID
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register ID

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails

- *data_specification.exceptions.ParameterOutOfBoundsException* –
  - If operand_1_is_register is True and operand_1 is not a valid register ID
  - If operand_2_is_register is True and operand_2 is not a valid register ID

logical_not(*register_id*, *operand*, *operand_is_register=False*)
Insert command to perform a logical xor operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The ID of the register to store the result in
- **operand** (*int*) –
  - If operand_is_register is True, the ID of a register where the first operand can be found, between 0 and 15
  - If operand_is_register is False, a 32-bit value
- **operand_is_register** (*bool*) – Indicates if operand_1 is a register ID

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- *data_specification.exceptions.ParameterOutOfBoundsException* –
  - If operand_is_register is True and operand_1 is not a valid register ID

logical_or(*register_id*, *operand_1*, *operand_2*, *operand_1_is_register=False*, *operand_2_is_register=False*)
Insert command to perform a logical OR operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The ID of the register to store the result in
- **operand_1** (*int*) –
  - If operand_1_is_register is True, the ID of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operand_2** (*int*) –
  - If operand_2_is_register is True, the ID of a register where the second operand can be found. between 0 and 15
  - If operand_2_is_register is False, a 32-bit value
- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register ID
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register ID

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised

- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If operand_1_is_register is True and operand_1 is not a valid register ID
  - If operand_2_is_register is True and operand_2 is not a valid register ID

logical_right_shift(*register_id*, *operand_1*, *operand_2*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform a logical right shift operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The ID of the register to store the result in
- **operand_1** (*int*) –
  - If operand_1_is_register is True, the ID of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operand_2** (*int*) –
  - If operand_2_is_register is True, the ID of a register where the second operand can be found. between 0 and 15
  - If operand_2_is_register is False, a 32-bit value
- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register ID
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register ID

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If operand_1_is_register is True and operand_1 is not a valid register ID
  - If operand_2_is_register is True and operand_2 is not a valid register ID

logical_xor(*register_id*, *operand_1*, *operand_2*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform a logical xor operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The ID of the register to store the result in
- **operand_1** (*int*) –
  - If operand_1_is_register is True, the ID of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operand_2** (*int*) –

- If operand_2_is_register is True, the ID of a register where the second operand can be found. between 0 and 15
- If operand_2_is_register is False, a 32-bit value

- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register ID
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register ID

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If operand_1_is_register is True and operand_1 is not a valid register ID
  - If operand_2_is_register is True and operand_2 is not a valid register ID

mem_slots

no_operation()
Insert command to execute nothing

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails

ongoing_function_definition

ongoing_loop

print_struct(*structure_id*, *structure_id_is_register=False*)
Insert command to print out a structure (for debugging)

Parameters

- **structure_id** (*int*) –
  - If structure_id_is_register is True, the ID of the register containing the ID of the structure to print, between 0 and 15
  - If structure_id_is_register is False, the ID of the structure to print, between 0 and 15
- **structure_id_is_register** (*bool*) – Indicates if the structure_id is a register

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised

- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If structure_id_is_register is True and structure_id is not a valid register ID
  - If structure_id_is_register is False and structure_id is not a valid structure ID
- ***data_specification.exceptions.NotAllocatedException*** – If structure_id_is_register is False and structure_id is the ID of a structure that has not been allocated

print_text(*text*, *encoding='ASCII'*)
Insert command to print some text (for debugging)

Parameters

- **text** (*str*) – The text to write (max 12 *bytes* once encoded)
- **encoding** – The character encoding to use for the string. Defaults to ASCII.

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails

print_value(*value*, *value_is_register=False*, *data_type=<DataType.UINT32: 2>*)
Insert command to print out a value (for debugging)

Parameters

- **value** (*float*) –
  - If value_is_register is True, the ID of the register containing the value to print
  - If value_is_register is False, the value to print as a float
- **value_is_register** (*bool*) – Indicates if the value is a register
- **data_type** (DataType) – The type of the data to be printed

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If value_is_register is True and value is not a valid register ID
  - If value_is_register is False, the data_type is an integer type and value has a fractional part
  - If value_is_register is False and the value would overflow the data type

- *data_specification.exceptions.UnknownTypeException* –
  - If data_type is not a valid data type

random_distribution

read_value(*dest_id*, *data_type*)
Insert command to read a value from the current write pointer, causing the write pointer to move by the number of bytes read. The data is stored in a register passed as argument.

Parameters

- **dest_id** (*int*) – The ID of the destination register.
- **data_type** (DataType) – The type of the data to be read.

Returns Nothing is returned

Return type None

region_sizes
A list of sizes of each region that has been reserved. Note that the list will include 0s for each non-reserved region.

report_writer

reserve_memory_region(*region*, *size*, *label=None*, *empty=False*, *shrink=True*)
Insert command to reserve a memory region

Parameters

- **region** (*int*) – The number of the region to reserve, from 0 to 15
- **size** (*int*) – The size to reserve for the region in bytes
- **label** (*str*) – An optional label for the region
- **empty** (*bool*) – Specifies if the region will be left empty
- **shrink** (*bool*) – Specifies if the region will be compressed

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- *data_specification.exceptions.RegionInUseException* – If the region was already reserved
- *data_specification.exceptions.ParameterOutOfBoundsException* – If the region requested was out of the allowed range, or that the size was too big to fit in SDRAM

rng

save_write_pointer(*register_id*)
Insert command to save the write pointer to a register

Parameters **register_id** (*int*) – The ID of the register to assign, between 0 and 15

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** – If the register_id is not a valid register ID
- ***data_specification.exceptions.NoRegionSelectedException*** – If no region has been selected

set_register_value(*register_id*, *data*, *data_is_register=False*, *data_type=<DataType.UINT32: 2>*)

Insert command to set the value of a register

Parameters

- **register_id** (*int*) – The ID of the register to assign, between 0 and 15
- **data** (*float*) –
  - If data_is_register is True, the register ID containing the data to set, between 0 and 15
  - If data_is_register is False, the data as a float
- **data_is_register** (*bool*) – Indicates if data is a register ID
- **data_type** (DataType) – The type of the data to be assigned

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If register_id is not a valid register_id
  - If data_is_register is True, and data is not a valid register ID
  - If data_is_register is False, data_type is an integer type and data has a fractional part
  - If data_is_register if False, and data would overflow the data type
- ***data_specification.exceptions.UnknownTypeException*** – If the data type is not known

set_structure_value(*structure_id*, *parameter_index*, *value*, *data_type*, *value_is_register=False*)

Insert command to set a value in a structure

Parameters

- **structure_id** (*int*) –
  - If called outside of a function, the ID of the structure, between 0 and 15

  - If called within a function, the ID of the structure argument, between 0 and 4
- **parameter_index** (*int*) – The index of the value to assign in the structure, between 0 and the number of parameters declared in the structure
- **value** (*float*) –
  - If value_is_register is False, the value to assign at the selected position as a float
  - If value_is_register is True, the ID of the register containing the value to assign to the position, between 0 and 15
- **data_type** (DataType) – type of the data to be stored in the structure. If parameter value_is_register is set to true, this variable is disregarded
- **value_is_register** (*bool*) – Identifies if value identifies a register

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If structure_id is not in the allowed range
  - If parameter_index is larger than the number of parameters declared in the original structure
  - If value_is_register is False, and the data type of the structure value is an integer, and the value has a fractional part
  - If value_is_register is False, and value would overflow the position in the structure
  - If value_is_register is True, and value is not a valid register ID
- ***data_specification.exceptions.NotAllocatedException*** – If the structure requested has not been declared

set_write_pointer(*address*, *address_is_register=False*, *relative_to_current=False*)

Insert command to set the position of the write pointer within the current region

Parameters

- **address** (*int*) –
  - If address_is_register is True, the ID of the register containing the address to move to
  - If address_is_register is False, the address to move the write pointer to
- **address_is_register** (*bool*) – Indicates if the the address is a register ID
- **relative_to_current** (*bool*) – Indicates if the address is to be added to the current address, or used as an absolute address from the start of the current region

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** – If the address_is_register is True and address is not a valid register ID
- ***data_specification.exceptions.NoRegionSelectedException*** – If no region has been selected

spec_writer

start_conditional(*register_id*, *condition*, *value*, *value_is_register=False*)

Insert command to start a conditional if…then…else construct. If the condition evaluates to true, the statement is executed up to the next else statement, or the end of the conditional, whichever comes first.

Parameters

- **register_id** (*int*) – The ID of a register to compare the value of
- **condition** (Condition) – The condition which must be true to execute the following commands
- **value** (*int*) –
  - If value_is_register is False, the value to compare to the value in the register
  - If value_is_register is True, the ID of the register containing the value to compare, between 0 and 15
- **value_is_register** (*bool*) – Indicates if value is a register ID

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If the register_id is not a valid register ID
  - if value_is_register is True and value is not a valid register ID
- ***data_specification.exceptions.UnknownTypeException*** – If the condition is not a valid condition

start_function(*function_id*, *argument_by_value*)

Insert command to start a function definition, with up to 5 arguments, which are the IDs of structures to be used within the function, each of which can be passed by reference or by value. In the commands following this command up to the end_function command, structures can only be referenced using the numbers 1 to 5 which address the arguments, rather than the original structure IDs

Parameters

- **function_id** (*int*) – The ID of the function currently defined.

- **argument_by_value** (*list of bool*) – A list of up to 5 booleans indicating if the structure to be passed as an argument is to be passed by reference (i.e., changes made within the function are persisted) or by value (i.e., changes made within the function are lost when the function exits. The number of arguments is determined by the length of this list.

Returns The ID of the function, between 0 and 31

Return type int

Raises

- ***data_specification.exceptions.ParameterOutOfBoundsException*** – If there are too many items in the list of arguments
- ***data_specification.exceptions.InvalidCommandException*** – If there is already a function being defined at this point
- ***data_specification.exceptions.FunctionInUseException*** – If the function is already defined

start_loop(*counter_register_id*, *start*, *end*, *increment=1*, *start_is_register=False*, *end_is_register=False*, *increment_is_register=False*)

Insert command to start a loop

Parameters

- **counter_register_id** (*int*) – The ID of the register to use as the loop counter, between 0 and 15
- **start** (*int*) –
  - If start_is_register is False, the number at which to start the loop counter, >= 0
  - If start_is_register is True, the ID of the register containing the number at which to start the loop counter, between 0 and 15
- **end** (*int*) –
  - If end_is_register is False, the number which the loop counter must reach to stop the loop i.e. the loop will run while the contents of counter_register < end, >= 0
  - If end_is_register is True, the ID of the register containing the number at which to stop the loop, between 0 and 15
- **increment** (*int*) –
  - If increment_is_register is False, the amount by which to increment the loop counter on each run of the loop, >= 0
  - If increment_is_register is True, the ID of the register containing the amount by which to increment the loop counter on each run of the loop, between 0 and 15
- **start_is_register** (*bool*) – Indicates if start is a register ID
- **end_is_register** (*bool*) – Indicates if end is a register ID
- **increment_is_register** (*bool*) – Indicates if increment is a register ID

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised

- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If counter_register_id is not a valid register ID
  - If start_is_register is True and increment is not a valid register_id
  - If end_is_register is True and increment is not a valid register_id
  - If increment_is_register is True, and increment is not a valid register_id
  - If start_is_register is False and start is not in the allowed range
  - If end_is_register is False and end is not in the allowed range
  - If increment_is_register is False and increment is not in the allowed range

struct_slots

switch_write_focus(*region*)

Insert command to switch the region being written to

Parameters **region** (*int*) – The ID of the region to switch to, between 0 and 15

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** – If the region identifier is not valid
- ***data_specification.exceptions.NotAllocatedException*** – If the region has not been allocated
- ***data_specification.exceptions.RegionUnfilledException*** – If the selected region should not be filled

txt_indent

write_array(*array_values*, *data_type=<DataType.UINT32: 2>*)

Insert command to write an array, causing the write pointer to move on by (data type size * the array size), in bytes.

Parameters

- **array_values** (*list of unsigned int*) – An array of words to be written
- **data_type** (*DataType*) – Type of data contained in the array

Returns The position of the write pointer within the current region, in bytes from the start of the region

Return type int

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.NoRegionSelectedException*** – If no region has been previously selected

write_cmd(*cmd_word_list*, *cmd_string*)

Applies write commands created previously created (and cached).

Note: See *create_cmd()* for how to create the arguments to this method.

Parameters

- **cmd_word_list** (*bytearray*) – list of binary words to be added to the binary data specification file
- **cmd_string** (*str*) – string describing the command to be added to the report for the data specification file

Returns Nothing is returned

Return type None

Raises

- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.NoRegionSelectedException*** – If no region has been selected to write to

write_command_to_files(*cmd_word_list*, *cmd_string*, *indent=False*, *outdent=False*, *no_instruction_number=False*)

Writes the binary command to the binary output file and, if the user has requested a text output for debug purposes, also write the text version to the text file. Setting the optional parameter 'indent' to True causes subsequent commands to be indented by two spaces relative to this one. Similarly, setting 'outdent' to True, reverses this spacing.

Parameters

- **cmd_word_list** (*bytearray*) – list of binary words to be added to the binary data specification file
- **cmd_string** (*str*) – string describing the command to be added to the report for the data specification file
- **indent** (*bool*) – if the following lines need to be indented
- **outdent** (*bool*) – if the following lines need to be out-dented
- **no_instruction_number** – if each report line should include also the address of the command in the file
- **no_instruction_number** – bool

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails

write_repeated_value(*data*, *repeats=1*, *repeats_is_register=False*, *data_type=<DataType.UINT32: 2>*)

Insert command to write a value one or more times to the current write pointer, causing the write pointer to move on by the number of bytes required to represent the data type. The data is passed as a parameter to this function

Parameters

- **data** (*float*) – the data to write as a float.
- **repeats** (*int*) –
  - If repeats_is_register is False, this parameter identifies the number of times to repeat the data, between 1 and 255 (default 1)
  - If repeats_is_register is True, this parameter identifies the register that contains the number of repeats.
- **repeats_is_register** (*bool*) – Indicates if the parameter repeats identifies the register containing the number of repeats of the value to write
- **data_type** (DataType) – the type to convert data to

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If repeats_register is None, and repeats is out of range
  - If repeats_register is not a valid register ID
  - If data_type is an integer type, and data has a fractional part
  - If data would overflow the data type
- ***data_specification.exceptions.UnknownTypeException*** – If the data type is not known
- ***data_specification.exceptions.InvalidSizeException*** – If the data size is invalid
- ***data_specification.exceptions.NoRegionSelectedException*** – If no region has been selected to write to

write_structure(*structure_id*, *repeats=1*, *repeats_is_register=False*)

Insert command to write a structure to the current write pointer, causing the current write pointer to move on by the number of bytes needed to represent the structure

Parameters

- **structure_id** (*int*) –
  - If called within a function, the ID of the structure to write, between 0 and 15
  - If called outside of a function, the ID of the structure argument, between 0 and 5
- **repeats** (*int*) –
  - If repeats_is_register is True, the ID of the register containing the number of repeats, between 0 and 15
  - If repeats_is_register is False, the number of repeats to write, between 0 and 255
- **repeats_is_register** (*bool*) – Identifies if repeats identifies a register

Returns No value returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If repeats_is_register is False and structure_id is not a valid ID
  - If repeats_is_register is True and structure_id
  - If the number of repeats is out of range
- ***data_specification.exceptions.NoRegionSelectedException*** – If no region has been selected to write to
- ***data_specification.exceptions.RegionExhaustedException*** – If the selected region has no more space

write_value(*data*, *data_type=<DataType.UINT32: 2>*)

Insert command to write a value one or more times to the current write pointer, causing the write pointer to move on by the number of bytes required to represent the data type. The data is passed as a parameter to this function

---

Note: This method used to have two extra parameters repeats and repeats_is_register. They have been removed here. If you need them, use write_repeated_value

---

Parameters

- **data** (*float*) – the data to write as a float.
- **data_type** (DataType) – the type to convert data to

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised

- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If repeats_register is None, and repeats is out of range
  - If repeats_register is not a valid register ID
  - If data_type is an integer type, and data has a fractional part
  - If data would overflow the data type
- ***data_specification.exceptions.UnknownTypeException*** – If the data type is not known
- ***data_specification.exceptions.InvalidSizeException*** – If the data size is invalid
- ***data_specification.exceptions.NoRegionSelectedException*** – If no region has been selected to write to

write_value_from_register(*data_register*, *repeats=1*, *repeats_is_register=False*, *data_type=<DataType.UINT32: 2>*)

Insert command to write a value one or more times at the write pointer of the current memory region, causing it to move. The data is contained in a register whose ID is passed to the function

Parameters

- **data_register** (*int*) – Identifies the register in which the data is stored.
- **repeats** (*int*) –
  - If repeats_register is None, this parameter identifies the number of times to repeat the data, between 1 and 255 (default 1)
  - If repeats_register is not None (i.e. has an integer value), the content of this parameter is disregarded
- **repeats_is_register** (*None or int*) – Identifies the register containing the number of repeats of the value to write
- **data_type** (DataType) – the type of the data held in the register

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If repeats_register is None, and repeats is out of range
  - If repeats_register is not a valid register ID
  - If data_register is not a valid register ID

- ***data_specification.exceptions.UnknownTypeException*** – If the data type is not known
- ***data_specification.exceptions.NoRegionSelectedException*** – If no region has been selected to write to
- ***data_specification.exceptions.RegionExhaustedException*** – If the selected region has no more space

3.1.8 data_specification.exceptions module

exception data_specification.exceptions.**DataSpecificationException**
Bases: exceptions.Exception

A general purpose exception indicating that something went wrong when interacting with a Data Specification

exception data_specification.exceptions.**DataSpecificationSyntaxError**
Bases: *data_specification.exceptions.DataSpecificationException*

An exception which occurs when a command read from the data specification file shows an inconsistency in the binary content.

exception data_specification.exceptions.**DataUndefinedWriterException**
Bases: exceptions.Exception

An exception that indicates that the file data writer has not been initialised

exception data_specification.exceptions.**DuplicateParameterException**(*command*, *parameters*)
Bases: *data_specification.exceptions.DataSpecificationException*

And exception that indicates that a command has been called with a duplicate parameter, which shouldn't be allowed.

Parameters

- **command** (*int*) – The command called with duplicate parameters
- **parameters** (*list*) – The parameters used to call the function

exception data_specification.exceptions.**ExecuteBreakInstruction**(*address*, *filename*)
Bases: *data_specification.exceptions.DataSpecificationException*

An exception which occurs when a BREAK instruction is found in the data specification.

Parameters

- **address** (*int*) – address of the data specification being executed at the time of breakpoint
- **filename** – file being parsed
- **filename** – str

exception data_specification.exceptions.**FunctionInUseException**(*function_id*)
Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a function is already defined

Parameters **function_id** (*int*) – The ID of the function

exception data_specification.exceptions.**InvalidCommandException**(*command*)
Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that the command being requested cannot be executed at this point in the specification.

Parameters **command** (*str*) – The command being executed

exception data_specification.exceptions.**InvalidOperationException**(*operation_type*, *requested_operation_id*, *command*)
Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that the operation of the type given type is not available.

Parameters

- **operation_type** (*str*) – The type of operation requested (i.e. arithmetic or logic)
- **requested_operation_id** (*int*) – The ID of the requested operation
- **command** (*str*) – The command being executed

exception data_specification.exceptions.**InvalidSizeException**(*type_name*, *type_size*, *command*)
Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that the size of the requested type is invalid

Parameters

- **type_name** (*str*) – The name of the requested type
- **type_size** (*int*) – The size of the requested variable
- **command** (*str*) – The command being executed

exception data_specification.exceptions.**NestedFunctionException**
Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a function is being defined within the context of another function definition

exception data_specification.exceptions.**NoMoreException**(*space_available*, *space_required*, *region*)
Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that there is no more space for the requested item

Parameters

- **space_available** (*int*) – The space available in the region
- **space_required** (*int*) – The space requested by the write command

exception data_specification.exceptions.**NoRegionSelectedException**(*command*)
Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a memory region has not been selected

Parameters **command** (*str*) – The command being executed

exception data_specification.exceptions.**NotAllocatedException**(*item_type*, *item_id*, *command*)
Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that an item is being used that has not been allocated

Parameters

- **item_type** (*str*) – The type of the item being used
- **item_id** (*int*) – The ID of the item being used
- **command** (*str*) – The command being executed

exception data_specification.exceptions.**ParameterOutOfBoundsException**(*parameter*, *value*, *range_min*, *range_max*, *command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a parameter value was outside of the allowed bounds

Parameters

- **parameter** (*str*) – The parameter that is out of bounds
- **value** (*float*) – The value specified
- **range_min** (*float*) – The minimum allowed value
- **range_max** (*float*) – The maximum allowed value
- **command** (*str*) – The command being executed

exception data_specification.exceptions.**RNGInUseException**(*rng_id*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a random number generator is already defined

Parameters **rng_id** (*int*) – The ID of the rng

exception data_specification.exceptions.**RandomNumberDistributionInUseException**(*rng_id*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a random number distribution is already defined

Parameters **rng_id** (*int*) – The ID of the random number distribution

exception data_specification.exceptions.**RegionExhaustedException**(*region*, *region_size*, *allocated_size*, *command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a region has run out of memory whilst some data is being written

Parameters

- **region** (*int*) – The region that was being written to
- **region_size** (*int*) – The originally requested size of the region that has run out of space, in bytes
- **allocated_size** (*int*) – The amount of the originally requested space that has already been allocated, in bytes
- **command** (*str*) – The command being executed

exception data_specification.exceptions.**RegionInUseException**(*region*, *label=None*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a region has already been allocated

Parameters

- **region** (*int*) – The region that was already allocated
- **label** – What label is known for the region

exception data_specification.exceptions.**RegionNotAllocatedException**(*region*, *command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception which occurs when trying to write to an unallocated region of memory

Parameters

- **region** –
- **command** –

exception data_specification.exceptions.**RegionOutOfBoundsException**(*region*, *region_size*, *requested_offset*, *command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that an offset into a region is out of bounds for that region

Parameters

- **region** (*int*) – The region that was being offset into
- **region_size** (*int*) – The originally requested size of the region in question, in bytes
- **requested_offset** (*int*) – The offset being requested, in bytes
- **command** (*str*) – The command being executed

exception data_specification.exceptions.**RegionUnfilledException**(*region*, *command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a memory region is being used that was originally requested to be unfilled

Parameters

- **region** (*int*) – The region that was requested as unfilled
- **command** (*str*) – The command being executed

exception data_specification.exceptions.**StructureInUseException**(*structure*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a structure has already been defined

Parameters **structure** (*int*) – The structure that was already allocated

exception data_specification.exceptions.**TablePointerOutOfMemoryException**(*memory_available*, *memory_required*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception which occurs when building the table pointer as header of the data generated by the spec executor. This message is printed in case the memory available is not enough to contain the pointer table for each of the allocated regions.

Parameters

- **memory_available** (*int*) – on-chip memory available
- **memory_required** (*int*) – on-chip memory required to complete the execution of the specification file

exception data_specification.exceptions.**TypeMismatchException**(*command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a type mismatch has occurred

Parameters **command** (*int*) – The command that generated the exception

exception data_specification.exceptions.**UnimplementedDSECommandError**(*command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception which occurs when trying to execute an unimplemented command

Parameters **command** (*str*) – Command attempted to be executed by the DSE

exception data_specification.exceptions.**UnimplementedDSGCommandError**(*command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception which occurs when trying to write an unimplemented command

Parameters **command** (*str*) – Command attempted to be generated by the DSG

exception data_specification.exceptions.**UnknownConditionException**(*condition_id*, *command*)

Bases: *data_specification.exceptions.DataSpecificationException*

And exception which is triggered in case the condition in an IF test does not exist in the list of possible conditions

Parameters

- **condition_id** (*int*) – ID of the condition being requested
- **command** (*str*) – The command being executed

exception data_specification.exceptions.**UnknownTypeException**(*type_id*, *command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that the value of the requested type is unknown

Parameters

- **type_id** (*int*) – The ID of the requested type
- **command** (*str*) – The command being executed

exception data_specification.exceptions.**UnknownTypeLengthException**(*data_length*, *command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that the value of the requested type is unknown

Parameters

- **data_length** (*int*) – the length of the requested type
- **command** (*str*) – The command being executed

exception data_specification.exceptions.**WrongParameterNumberException**(*function_id*, *no_of_parameters_required*, *parameters*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a function has been called with a wrong number of parameters.

Parameters

- **function_id** (*int*) – The ID of the function
- **parameters** (*list*) – The parameters used in the function call
- **no_of_parameters_required** – The number of parameters required by the function

3.1.9 data_specification.memory_region module

class data_specification.memory_region.**MemoryRegion**(*unfilled*, *size*)

Bases: object

Memory region storage object

Parameters **unfilled** – if the region needs to be filled when written

Return type None

Raises **None** – this method does not raise any known exception

allocated_size

The size of the region

Returns the size of the region

Return type int

Raises **None** – this method does not raise any known exception

increment_write_pointer(*n_bytes*)

max_write_pointer

region_data

The container which holds the data written in this region

Returns the region data container

Return type bytearray

Raises **None** – this method does not raise any known exception

remaining_space

The space between the current write pointer and the end of the/ region

Returns the number of bytes in the region that can be written

Return type int

Raises **None** – this method does not raise any known exception

unfilled

Whether the region is marked as not fillable

Returns True if the region will not contain any data at write time

Return type bool

Raises **None** – this method does not raise any known exception

write_pointer

3.1.10 data_specification.memory_region_collection module

class data_specification.memory_region_collection.**MemoryRegionCollection**(*n_regions*)

Bases: object

Collection of memory regions.

Create a new MemoryRegionCollection with the given number of regions.

count_used_regions()

Return type int

is_empty(*region*)

Return type bool

is_unfilled(*region*)

Return type bool

needs_to_write_region(*region*)

Helper method which determines if a unfilled region actually needs to be written (optimisation to stop large data files).

Parameters **region** – the region ID to which the test is being ran on

Returns a boolean stating if the region needs to be written

Return type bool

Raises ***NoRegionSelectedException*** – when the ID is beyond the expected region range

regions

Return type iterable(*MemoryRegion*)

3.1.11 data_specification.utility_calls module

Utility calls for interpreting bits of the DSG

data_specification.utility_calls.**get_data_spec_and_file_writer_filename**(*processor_chip_x*, *processor_chip_y*, *processor_id*, *hostname*, *report_directory='TEMP'*, *write_text_specs=False*, *application_run_time_report_folde*

Encapsulates the creation of the DSG writer and the file paths :param processor_chip_x: x-coordinate of the chip :type processor_chip_x: int :param processor_chip_y: y-coordinate of the chip :type processor_chip_y: int :param processor_id: The processor ID :type processor_id: int :param hostname: The hostname of the SpiNNaker machine :type hostname: str :param report_directory: the directory for the reports folder :type report_directory: file path :param write_text_specs: True if a textual version of the specification should be written :type write_text_specs: bool :param application_run_time_report_folder: The folder to contain the resulting specification files; if 'TEMP' then a temporary directory is used. :type application_run_time_report_folder: str :return: the filename of the data writer and the data specification object :rtype: tuple(str, *DataSpecificationGenerator*)

data_specification.utility_calls.**get_region_base_address_offset**(*app_data_base_address*, *region*)

Find the address of the of a given region for the DSG :param app_data_base_address: base address for the core :param region: the region ID we're looking for

data_specification.utility_calls.**get_report_writer**(*processor_chip_x*, *processor_chip_y*, *processor_id*, *hostname*, *report_directory='TEMP'*, *write_text_specs=False*)

check if text reports are needed and if so initialise the report writer to send down to DSG

Parameters

- **processor_chip_x** (*int*) – x-coordinate of the chip
- **processor_chip_y** (*int*) – y-coordinate of the chip
- **processor_id** (*int*) – The processor ID
- **hostname** (*str*) – The hostname of the SpiNNaker machine
- **report_directory** (*file path*) – the directory for the reports folder
- **write_text_specs** (*bool*) – True if a textual version of the specification should be written

Returns the report_writer_object or None if not reporting

3.1.12 Module contents

Used to generate memory images for a SpiNNaker CPU core from a set of instructions.

The main part of this package is the *data_specification.DataSpecificationGenerator* class. This is used to generate a "Data Specification", which can then be executed to produce a memory image. This package also handles this function if required, through the *data_specification.DataSpecificationExecutor* class.

3.1.12.1 Functional Requirements

Creation of a Data Specification Language file which can be executed to produce a memory image.

- Any errors that can be checked during the creation of the specification should throw an exception.
- It will be impossible to detect all errors at creation time.
- There should be no assumption of where the data specification will be stored, although a default provision of a way to write the specification to a file is acceptable.

Execution of a Data Specification Language file, producing a memory image.

- This should detect any errors during execution and report them, halting the execution.
- There should be no assumption of where the data specification is read from, although a default provision of a way to read the specification from a file is acceptable.

3.1.12.2 Use Cases

There are a number of use-cases of this library:

- *DataSpecificationGenerator* is used to create a compressed memory image which can be expanded later, to reduce the amount of data that needs to be transferred over a slow connection.
- *DataSpecificationExecutor* is used to execute a previously generated specification at the receiving end of a slow connection.

class data_specification.**DataSpecificationExecutor**(*spec_reader*, *memory_space*)
Bases: object

Used to execute a SpiNNaker data specification language file to produce a memory image

Parameters

- **spec_reader** (AbstractDataReader) – The object to read the specification language file from
- **memory_space** (*int*) – memory available on the destination architecture

Raises

- **spinn_storage_handlers.exceptions.DataReadException** – If a read from external storage fails
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails

dsef

execute()
Executes the specification

Returns Nothing

Raises

- **spinn_storage_handlers.exceptions.DataReadException** – If a read from external storage fails

- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- *data_specification.exceptions.DataSpecificationException* – If there is an error when executing the specification
- **data_specification.exceptions.TablePointerOutOfMemory** – If the table pointer generated as data header exceeds the size of the available memory

get_constructed_data_size()
Return the size of the data that will be written to memory

Returns size of the data that will be written to memory

Return type int

get_header()
Get the header of the data as a numpy array

get_pointer_table(*start_address*)
Get the pointer table as a numpy array

Parameters **start_address** – The base address of the data to be written

Return type numpy.array

get_region(*region_id*)
Get a region with a given ID

Parameters **region_id** (*int*) – The ID of the region to get

Returns The region, or None if the region was not allocated

Return type *MemoryRegion*

spec_reader

class data_specification.**DataSpecificationExecutorFunctions**(*spec_reader*, *memory_space*)
Bases: *data_specification.abstract_executor_functions.AbstractExecutorFunctions*

This class includes the function related to each of the commands of the data specification file.

Parameters

- **spec_reader** (AbstractDataReader) – The object to read the specification language file from
- **memory_space** (*int*) – Memory space available for the data to be generated

cmd_size

current_region

data_len

dest_reg

execute_break(*cmd*)
This command raises an exception to stop the execution of the data spec executor (DSE)

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

Raises *data_specification.exceptions.ExecuteBreakInstruction* – Raises the exception to break the execution of the DSE

execute_end_spec(*cmd*)
Return the value which terminates the data spec executor

Parameters **cmd** (*int*) – the command which triggered the function call

Returns END_SPEC_EXECUTOR

Return type int

Raises *data_specification.exceptions.DataSpecificationSyntaxError* – If command END_SPEC != -1

execute_mv(*cmd*)
This command moves an immediate value to a register or copies the value of a register to another register

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

Raises *data_specification.exceptions.DataSpecificationSyntaxError* – If the destination register is not correctly specified; the destination must be a register and the appropriate bit needs to be set in the specification

execute_reserve(*cmd*)
This command reserves a region and assigns some memory space to it

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

Raises

- *data_specification.exceptions.DataSpecificationSyntaxError* – If there is an error in the command syntax
- *data_specification.exceptions.ParameterOutOfBoundsException* – If the requested size of the region is beyond the available memory space

execute_set_wr_ptr(*cmd*)

execute_switch_focus(*cmd*)
This command switches the focus to the desired, already allocated, memory region

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

Raises *data_specification.exceptions.RegionUnfilledException* – If the focus is being switched to a region of memory which has been declared to be kept unfilled

execute_write(*cmd*)
This command writes the given value in the specified region a number of times as identified by either a value in the command or a register value

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

Raises *data_specification.exceptions.DataSpecificationSyntaxError* – If there is an error in the command syntax

execute_write_array(*cmd*)
This command writes an array of values in the specified region

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

mem_regions

memory_space

opcode

registers

space_allocated

spec_reader

src1_reg

src2_reg

class data_specification.**DataSpecificationGenerator**(*spec_writer*, *report_writer=None*)
Bases: object

Used to generate a SpiNNaker data specification language file that can be executed to produce a memory image

Parameters

- **spec_writer** (AbstractDataWriter) – The object to write the specification to
- **report_writer** (AbstractDataWriter) – Determines if a text version of the specification is to be written and, if so, where. No report is written if this is None.

Raises **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails

MAGIC_NUMBER = 2903706326

VERSION = 1

align_write_pointer(*log_block_size*, *log_block_size_is_register=False*, *return_register_id=None*)
Insert command to align the write pointer against a power-of-2 block size in bytes. Zeros are inserted in the intervening space

Parameters

- **log_block_size** (*int*) –
  - If log_block_size_is_register is False, log to base 2 of the block size (e.g. The write pointer will be moved so that it is a multiple of 2^(log_block_size)), between 0 and 32
  - If log_block_size_is_register is True, the ID of the register containing log to the base 2 of the block size, between 0 and 15
- **log_block_size_is_register** (*bool*) – Indicates if log_block_size is a register ID

- **return_register_id** (*int*) – The ID of a register where the write pointer will be written to once it has been updated, between 0 and 15or None if no such writing is to be done

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If log_block_size_is_register is False, and log_block_size is not within the allowed range
  - If log_block_size_is_register is True and log_block_size is not a valid register ID
- ***data_specification.exceptions.RegionOutOfBoundsException*** – If the move of the pointer would put it outside of the current region
- ***data_specification.exceptions.NoRegionSelectedException*** – If no region has been selected

break_loop()

Insert command to break out of a loop before it has completed

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.InvalidCommandException*** – If there is no loop in operation at this point

call_arithmetic_operation(*register_id*, *operand_1*, *operation*, *operand_2*, *signed*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform an arithmetic operation on two signed or unsigned values and store the result in a register

Parameters

- **register_id** (*int*) – The ID of the register to store the result in
- **operand_1** (*int*) –
  - If operand_1_is_register is True, the ID of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operation** (ArithmeticOperation) – The operation to perform
- **operand_2** (*int*) –

- If operand_2_is_register is True, the ID of a register where the second operand can be found, between 0 and 15
- If operand_2_is_register is False, a 32-bit value

- **signed** (*bool*) – Indicates if the values should be considered signed
- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register ID
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register ID

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If operand_1_is_register is True and operand_1 is not a valid register ID
  - If operand_2_is_register is True and operand_2 is not a valid register ID
- ***data_specification.exceptions.UnknownTypeException*** – If operation is not a known operation

call_function (*function_id*, *structure_ids*)

Insert command to call a function

Parameters

- **function_id** (*int*) – The ID of a previously defined function, between 0 and 31
- **structure_ids** (*list of int*) – A list of structure_ids that will be passed into the function, each between 0 and 15

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If the function ID is not valid
  - If any of the structure IDs are not valid
- ***data_specification.exceptions.NotAllocatedException*** –
  - If a function has not been defined with the given ID
  - If no structure has been defined with one of the IDs in structure_ids
- ***data_specification.exceptions.WrongParameterNumberException*** – If a function is called with a wrong number of parameters

- ***data_specification.exceptions.DuplicateParameterException*** – If a function is called with duplicate parameters

call_random_distribution(*distribution_id*, *register_id*)

Insert command to get the next random number from a random distribution, placing the result in a register to be used in a future call

Parameters

- **distribution_id** (*int*) – The ID of the random distribution to call between 0 and 63
- **register_id** (*int*) – The ID of the register to store the result in between 0 and 15

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.NotAllocatedException*** – If the random distribution ID was not previously declared
- ***data_specification.exceptions.ParameterOutOfBoundsException*** – If the distribution_id or register_id specified was out of range

comment(*comment*)

Write a comment to the text version of the specification. Note that this is ignored by the binary file

Parameters **comment** (*str*) – The comment to write

Returns Nothing is returned

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails

conditionals

copy_structure(*source_structure_id*, *destination_structure_id*, *source_id_is_register=False*, *destination_id_is_register=False*)

Insert command to copy a structure, possibly overwriting another structure

Parameters

- **source_structure_id** (*int*) –
  - If source_id_is_register is True, the ID of the register holding the source structure ID, between 0 and 15
  - Otherwise, the ID of the source structure, between 0 and 15
- **destination_structure_id** (*int*) –
  - If destination_id_is_register is True, the ID of the register holding the destination structure ID, between 0 and 15

- If destination_id_is_register is False, the ID of the destination structure, between 0 and 15

- **source_id_is_register** (*bool*) – Indicates if source_structure_id is a register ID
- **destination_id_is_register** (*bool*) – Indicates if destination_structure_id is a register ID

Returns The ID of the copied structure, between 0 and 15

Return type int

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If source_id_is_register is True and source_structure_id is not a valid register ID
  - If destination_id_is_register is True and destination_structure_id is not a valid register ID
  - If source_id_is_register is False and source_structure_id is not a valid structure ID
  - If destination_id_is_register is False and destination_structure_id is not a valid structure ID
- ***data_specification.exceptions.NotAllocatedException*** –
  - If no structure with ID source_structure_id has been allocated

copy_structure_parameter(*source_structure_id*, *source_parameter_index*, *destination_id*, *destination_parameter_index=None*, *destination_is_register=False*)

Insert command to copy the value of a parameter from one structure to another.

Parameters

- **source_structure_id** (*int*) – The ID of the source structure, between 0 and 15
- **source_parameter_index** (*int*) – The index of the parameter in the source structure
- **destination_id** (*int*) – The ID of the destination structure, or the ID of the destination register, between 0 and 15
- **destination_parameter_index** (*int*) – The index of the parameter in the destination structure. Ignored when writing to a register.
- **destination_is_register** (*bool*) – Indicates whether the destination is a structure or a register.

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised

- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- *data_specification.exceptions.ParameterOutOfBoundsException* –
  - If source_structure_id is not a valid structure ID
  - If destination_id is not a valid structure ID
  - If source_parameter_index is not a valid parameter index in the source structure
  - If destination_parameter_index is not a valid parameter index in the destination structure
- *data_specification.exceptions.NotAllocatedException* –
  - If no structure with ID destination_id has been allocated
  - If no structure with ID source_structure_id has been allocated

create_cmd(*data*, *data_type=<DataType.UINT32: 2>*)

Creates command to write a value to the current write pointer, causing the write pointer to move on by the number of bytes required to represent the data type. The data is passed as a parameter to this function.

Note: This does not actually insert the WRITE command in the spec; that is done by *write_cmd()*.

Parameters

- **data** (*int or float*) – the data to write.
- **data_type** (DataType) – the type to convert data to

Returns cmd_word_list; list of binary words to be added to the binary data specification file, and cmd_string; string describing the command to be added to the report for the data specification file

Return type (bytearray, str)

Raises

- *data_specification.exceptions.ParameterOutOfBoundsException* –
  - If repeats_register is None, and repeats is out of range
  - If repeats_register is not a valid register ID
  - If data_type is an integer type, and data has a fractional part
  - If data would overflow the data type
- *data_specification.exceptions.UnknownTypeException* – If the data type is not known
- *data_specification.exceptions.InvalidSizeException* – If the data size is invalid

current_region

declare_random_number_generator(*rng_id*, *rng_type*, *seed*)

Insert command to declare a random number generator

Parameters

- **rng_id** (*int*) – The ID of the random number generator
- **rng_type** (RandomNumberGenerator) – The type of the random number generator
- **seed** (*int*) – The seed of the random number generator >= 0

Returns The ID of the created random number generator, between 0 and 15

Return type int

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.UnknownTypeException*** – If the rng_type is not one of the allowed values
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If the seed is too big or too small
  - If the rng_id is not in the allowed range
- ***data_specification.exceptions.RNGInUseException*** – If the random number generator with the given ID has already been defined

declare_uniform_random_distribution(*distribution_id*, *structure_id*, *rng_id*, *min_value*, *max_value*)

Insert commands to declare a uniform random distribution

Parameters

- **distribution_id** – ID of the distribution being set up
- **distribution_id** – int
- **structure_id** (*int*) – ID of an empty structure slot to fill with the uniform random distribution data
- **rng_id** (*int*) – The ID of the random number generator, between 0 and 15
- **min_value** (*float*) – The minimum value that should be returned from the distribution between -32768.0 and max_value
- **max_value** (*float*) – The maximum value that should be returned from the distribution between min_value and 32767.9999847

Returns The ID of the created uniform random distribution to be used in future calls of the distribution between 0 and 63

Return type int

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.NoMoreException*** – If there is no more space for a new random distribution

- *data_specification.exceptions.NotAllocatedException* – If the requested rng_id has not been allocated
- *data_specification.exceptions.ParameterOutOfBoundsException* – If rng_id, structure_id, min_value or max_value is out of range
- *data_specification.exceptions.StructureInUseException* – If structure structure_id is already defined

define_break()
Insert command to stop execution with an exception (for debugging)

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails

define_structure(*structure_id*, *parameters*)
Insert commands to define a data structure

Parameters

- **structure_id** (*int*) – the ID of the structure to create, between 0 and 15
- **parameters** (list of (str, DataType, float)) – A list of between 1 and 255 tuples of (label, data_type, value) where: * label is the label of the element (for debugging) * data_type is the data type of the element * value is the value of the element, or None if no value is to be assigned

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- *data_specification.exceptions.NoMoreException* – If there are no more spaces for new structures
- *data_specification.exceptions.ParameterOutOfBoundsException* –
  - If there are an incorrect number of parameters
  - If the size of one of the tuples is incorrect
  - If one of the values to be assigned has an integer data_type but has a fractional part
  - If one of the values to be assigned would overflow its data_type
- *data_specification.exceptions.UnknownTypeException* – If one of the data types in the structure is unknown

else_conditional()
Insert command for the else of an if...then...else construct. If the condition of the conditional evaluates to false, the statements up between the conditional and the insertion of this "else" are skipped, and the statements from this point until the end of the conditional are executed.

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- *data_specification.exceptions.InvalidCommandException* – If there is no conditional in operation at this point

end_conditional()
Insert command to mark the end of an if...then...else construct

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- *data_specification.exceptions.InvalidCommandException* – If there is no conditional in operation at this point

end_function()
Insert command to mark the end of a function definition

Returns Nothing is returned

Return type None

Raises *data_specification.exceptions.InvalidCommandException* – If there is no function being defined at this point

end_loop()
Insert command to indicate that this is the end of the loop. Commands between the start of the loop and this command will be repeated.

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- *data_specification.exceptions.InvalidCommandException* – If there is no loop in operation at this point

end_specification(*close_writer=True*)
Insert a command to indicate that the specification has finished and finish writing

Parameters **close_writer** (*bool*) – Indicates whether to close the underlying writer(s)

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails

free_memory_region(*region*)
Insert command to free a previously reserved memory region

Parameters **region** (*int*) – The number of the region to free, from 0 to 15

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.NotAllocatedException*** – If the region was not reserved
- ***data_specification.exceptions.ParameterOutOfBoundsException*** – If the region requested was out of the allowed range

function_slots

get_structure_value(*destination_id*, *structure_id*, *parameter_index*, *parameter_index_is_register=False*)
Insert command to get a value from a structure. The value is copied in a register.

Parameters

- **destination_id** (*int*) – The ID of the destination register
- **structure_id** (*int*) – The ID of the source structure
- **parameter_index** (*int*) – The ID of the parameter/element to copy
- **parameter_index_is_register** (*bool*) – True if the index of the structure is contained in a register

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails

- *data_specification.exceptions.ParameterOutOfBoundsException* –
  - If structure_id is not in the allowed range
  - If parameter_index is larger than the number of parameters declared in the original structure
  - If destination_id is not the ID of a valid register
  - If parameter_index_is_register is True and parameter_index is not a valid register ID
- *data_specification.exceptions.NotAllocatedException* – If the structure requested has not been declared

instruction_counter

logical_and(*register_id*, *operand_1*, *operand_2*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform a logical AND operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The ID of the register to store the result in
- **operand_1** (*int*) –
  - If operand_1_is_register is True, the ID of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operand_2** (*int*) –
  - If operand_2_is_register is True, the ID of a register where the second operand can be found. between 0 and 15
  - If operand_2_is_register is False, a 32-bit value
- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register ID
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register ID

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- *data_specification.exceptions.ParameterOutOfBoundsException* –
  - If operand_1_is_register is True and operand_1 is not a valid register ID
  - If operand_2_is_register is True and operand_2 is not a valid register ID

logical_left_shift(*register_id*, *operand_1*, *operand_2*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform a logical left shift operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The ID of the register to store the result in
- **operand_1** (*int*) –

- If operand_1_is_register is True, the ID of a register where the first operand can be found, between 0 and 15
- If operand_1_is_register is False, a 32-bit value

- **operand_2** (*int*) –
  - If operand_2_is_register is True, the ID of a register where the second operand can be found. between 0 and 15
  - If operand_2_is_register is False, a 32-bit value
- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register ID
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register ID

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If operand_1_is_register is True and operand_1 is not a valid register ID
  - If operand_2_is_register is True and operand_2 is not a valid register ID

logical_not(*register_id*, *operand*, *operand_is_register=False*)

Insert command to perform a logical xor operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The ID of the register to store the result in
- **operand** (*int*) –
  - If operand_is_register is True, the ID of a register where the first operand can be found, between 0 and 15
  - If operand_is_register is False, a 32-bit value
- **operand_is_register** (*bool*) – Indicates if operand_1 is a register ID

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If operand_is_register is True and operand_1 is not a valid register ID

logical_or(*register_id*, *operand_1*, *operand_2*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform a logical OR operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The ID of the register to store the result in

- **operand_1** (*int*) –
  - If operand_1_is_register is True, the ID of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operand_2** (*int*) –
  - If operand_2_is_register is True, the ID of a register where the second operand can be found. between 0 and 15
  - If operand_2_is_register is False, a 32-bit value
- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register ID
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register ID

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If operand_1_is_register is True and operand_1 is not a valid register ID
  - If operand_2_is_register is True and operand_2 is not a valid register ID

logical_right_shift(*register_id*, *operand_1*, *operand_2*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform a logical right shift operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The ID of the register to store the result in
- **operand_1** (*int*) –
  - If operand_1_is_register is True, the ID of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operand_2** (*int*) –
  - If operand_2_is_register is True, the ID of a register where the second operand can be found. between 0 and 15
  - If operand_2_is_register is False, a 32-bit value
- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register ID
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register ID

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails

- *data_specification.exceptions.ParameterOutOfBoundsException* –
  - If operand_1_is_register is True and operand_1 is not a valid register ID
  - If operand_2_is_register is True and operand_2 is not a valid register ID

logical_xor(*register_id*, *operand_1*, *operand_2*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform a logical xor operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The ID of the register to store the result in
- **operand_1** (*int*) –
  - If operand_1_is_register is True, the ID of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operand_2** (*int*) –
  - If operand_2_is_register is True, the ID of a register where the second operand can be found. between 0 and 15
  - If operand_2_is_register is False, a 32-bit value
- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register ID
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register ID

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- *data_specification.exceptions.ParameterOutOfBoundsException* –
  - If operand_1_is_register is True and operand_1 is not a valid register ID
  - If operand_2_is_register is True and operand_2 is not a valid register ID

mem_slots

no_operation()

Insert command to execute nothing

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails

ongoing_function_definition

ongoing_loop

print_struct(*structure_id*, *structure_id_is_register=False*)
Insert command to print out a structure (for debugging)

Parameters

- **structure_id** (*int*) –
  - If structure_id_is_register is True, the ID of the register containing the ID of the structure to print, between 0 and 15
  - If structure_id_is_register is False, the ID of the structure to print, between 0 and 15
- **structure_id_is_register** (*bool*) – Indicates if the structure_id is a register

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If structure_id_is_register is True and structure_id is not a valid register ID
  - If structure_id_is_register is False and structure_id is not a valid structure ID
- ***data_specification.exceptions.NotAllocatedException*** – If structure_id_is_register is False and structure_id is the ID of a structure that has not been allocated

print_text(*text*, *encoding='ASCII'*)
Insert command to print some text (for debugging)

Parameters

- **text** (*str*) – The text to write (max 12 *bytes* once encoded)
- **encoding** – The character encoding to use for the string. Defaults to ASCII.

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails

print_value(*value*, *value_is_register=False*, *data_type=<DataType.UINT32: 2>*)
Insert command to print out a value (for debugging)

Parameters

- **value** (*float*) –
  - If value_is_register is True, the ID of the register containing the value to print
  - If value_is_register is False, the value to print as a float
- **value_is_register** (*bool*) – Indicates if the value is a register

- **data_type** (DataType) – The type of the data to be printed

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- *data_specification.exceptions.ParameterOutOfBoundsException* –
  - If value_is_register is True and value is not a valid register ID
  - If value_is_register is False, the data_type is an integer type and value has a fractional part
  - If value_is_register is False and the value would overflow the data type
- *data_specification.exceptions.UnknownTypeException* –
  - If data_type is not a valid data type

random_distribution

read_value(*dest_id*, *data_type*)

Insert command to read a value from the current write pointer, causing the write pointer to move by the number of bytes read. The data is stored in a register passed as argument.

Parameters

- **dest_id** (*int*) – The ID of the destination register.
- **data_type** (DataType) – The type of the data to be read.

Returns Nothing is returned

Return type None

region_sizes

A list of sizes of each region that has been reserved. Note that the list will include 0s for each non-reserved region.

report_writer

reserve_memory_region(*region*, *size*, *label=None*, *empty=False*, *shrink=True*)

Insert command to reserve a memory region

Parameters

- **region** (*int*) – The number of the region to reserve, from 0 to 15
- **size** (*int*) – The size to reserve for the region in bytes
- **label** (*str*) – An optional label for the region
- **empty** (*bool*) – Specifies if the region will be left empty
- **shrink** (*bool*) – Specifies if the region will be compressed

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.RegionInUseException*** – If the region was already reserved
- ***data_specification.exceptions.ParameterOutOfBoundsException*** – If the region requested was out of the allowed range, or that the size was too big to fit in SDRAM

rng

save_write_pointer(*register_id*)

Insert command to save the write pointer to a register

Parameters **register_id** (*int*) – The ID of the register to assign, between 0 and 15

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** – If the register_id is not a valid register ID
- ***data_specification.exceptions.NoRegionSelectedException*** – If no region has been selected

set_register_value(*register_id*, *data*, *data_is_register=False*, *data_type=<DataType.UINT32: 2>*)

Insert command to set the value of a register

Parameters

- **register_id** (*int*) – The ID of the register to assign, between 0 and 15
- **data** (*float*) –
  - If data_is_register is True, the register ID containing the data to set, between 0 and 15
  - If data_is_register is False, the data as a float
- **data_is_register** (*bool*) – Indicates if data is a register ID
- **data_type** (DataType) – The type of the data to be assigned

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised

- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If register_id is not a valid register_id
  - If data_is_register is True, and data is not a valid register ID
  - If data_is_register is False, data_type is an integer type and data has a fractional part
  - If data_is_register if False, and data would overflow the data type
- ***data_specification.exceptions.UnknownTypeException*** – If the data type is not known

set_structure_value(*structure_id*, *parameter_index*, *value*, *data_type*, *value_is_register=False*)

Insert command to set a value in a structure

Parameters

- **structure_id** (*int*) –
  - If called outside of a function, the ID of the structure, between 0 and 15
  - If called within a function, the ID of the structure argument, between 0 and 4
- **parameter_index** (*int*) – The index of the value to assign in the structure, between 0 and the number of parameters declared in the structure
- **value** (*float*) –
  - If value_is_register is False, the value to assign at the selected position as a float
  - If value_is_register is True, the ID of the register containing the value to assign to the position, between 0 and 15
- **data_type** (DataType) – type of the data to be stored in the structure. If parameter value_is_register is set to true, this variable is disregarded
- **value_is_register** (*bool*) – Identifies if value identifies a register

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If structure_id is not in the allowed range
  - If parameter_index is larger than the number of parameters declared in the original structure
  - If value_is_register is False, and the data type of the structure value is an integer, and the value has a fractional part
  - If value_is_register is False, and value would overflow the position in the structure

- If value_is_register is True, and value is not a valid register ID

- ***data_specification.exceptions.NotAllocatedException*** – If the structure requested has not been declared

set_write_pointer(*address*, *address_is_register=False*, *relative_to_current=False*)
Insert command to set the position of the write pointer within the current region

Parameters

- **address** (*int*) –
  - If address_is_register is True, the ID of the register containing the address to move to
  - If address_is_register is False, the address to move the write pointer to
- **address_is_register** (*bool*) – Indicates if the the address is a register ID
- **relative_to_current** (*bool*) – Indicates if the address is to be added to the current address, or used as an absolute address from the start of the current region

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** – If the address_is_register is True and address is not a valid register ID
- ***data_specification.exceptions.NoRegionSelectedException*** – If no region has been selected

spec_writer

start_conditional(*register_id*, *condition*, *value*, *value_is_register=False*)
Insert command to start a conditional if…then…else construct. If the condition evaluates to true, the statement is executed up to the next else statement, or the end of the conditional, whichever comes first.

Parameters

- **register_id** (*int*) – The ID of a register to compare the value of
- **condition** (Condition) – The condition which must be true to execute the following commands
- **value** (*int*) –
  - If value_is_register is False, the value to compare to the value in the register
  - If value_is_register is True, the ID of the register containing the value to compare, between 0 and 15
- **value_is_register** (*bool*) – Indicates if value is a register ID

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If the register_id is not a valid register ID
  - if value_is_register is True and value is not a valid register ID
- ***data_specification.exceptions.UnknownTypeException*** – If the condition is not a valid condition

start_function(*function_id*, *argument_by_value*)

Insert command to start a function definition, with up to 5 arguments, which are the IDs of structures to be used within the function, each of which can be passed by reference or by value. In the commands following this command up to the end_function command, structures can only be referenced using the numbers 1 to 5 which address the arguments, rather than the original structure IDs

Parameters

- **function_id** (*int*) – The ID of the function currently defined.
- **argument_by_value** (*list of bool*) – A list of up to 5 booleans indicating if the structure to be passed as an argument is to be passed by reference (i.e., changes made within the function are persisted) or by value (i.e., changes made within the function are lost when the function exits. The number of arguments is determined by the length of this list.

Returns The ID of the function, between 0 and 31

Return type int

Raises

- ***data_specification.exceptions.ParameterOutOfBoundsException*** – If there are too many items in the list of arguments
- ***data_specification.exceptions.InvalidCommandException*** – If there is already a function being defined at this point
- ***data_specification.exceptions.FunctionInUseException*** – If the function is already defined

start_loop(*counter_register_id*, *start*, *end*, *increment=1*, *start_is_register=False*, *end_is_register=False*, *increment_is_register=False*)

Insert command to start a loop

Parameters

- **counter_register_id** (*int*) – The ID of the register to use as the loop counter, between 0 and 15
- **start** (*int*) –
  - If start_is_register is False, the number at which to start the loop counter, >= 0
  - If start_is_register is True, the ID of the register containing the number at which to start the loop counter, between 0 and 15
- **end** (*int*) –

- If end_is_register is False, the number which the loop counter must reach to stop the loop i.e. the loop will run while the contents of counter_register < end, >= 0
- If end_is_register is True, the ID of the register containing the number at which to stop the loop, between 0 and 15

- **increment** (*int*) –
  - If increment_is_register is False, the amount by which to increment the loop counter on each run of the loop, >= 0
  - If increment_is_register is True, the ID of the register containing the amount by which to increment the loop counter on each run of the loop, between 0 and 15
- **start_is_register** (*bool*) – Indicates if start is a register ID
- **end_is_register** (*bool*) – Indicates if end is a register ID
- **increment_is_register** (*bool*) – Indicates if increment is a register ID

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If counter_register_id is not a valid register ID
  - If start_is_register is True and increment is not a valid register_id
  - If end_is_register is True and increment is not a valid register_id
  - If increment_is_register is True, and increment is not a valid register_id
  - If start_is_register is False and start is not in the allowed range
  - If end_is_register is False and end is not in the allowed range
  - If increment_is_register is False and increment is not in the allowed range

struct_slots

switch_write_focus (*region*)

Insert command to switch the region being written to

Parameters **region** (*int*) – The ID of the region to switch to, between 0 and 15

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails

- ***data_specification.exceptions.ParameterOutOfBoundsException*** – If the region identifier is not valid
- ***data_specification.exceptions.NotAllocatedException*** – If the region has not been allocated
- ***data_specification.exceptions.RegionUnfilledException*** – If the selected region should not be filled

txt_indent

write_array(*array_values*, *data_type=<DataType.UINT32: 2>*)

Insert command to write an array, causing the write pointer to move on by (data type size * the array size), in bytes.

Parameters

- **array_values** (*list of unsigned int*) – An array of words to be written
- **data_type** (*DataType*) – Type of data contained in the array

Returns The position of the write pointer within the current region, in bytes from the start of the region

Return type int

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.NoRegionSelectedException*** – If no region has been previously selected

write_cmd(*cmd_word_list*, *cmd_string*)

Applies write commands created previously created (and cached).

Note: See *create_cmd()* for how to create the arguments to this method.

Parameters

- **cmd_word_list** (*bytearray*) – list of binary words to be added to the binary data specification file
- **cmd_string** (*str*) – string describing the command to be added to the report for the data specification file

Returns Nothing is returned

Return type None

Raises

- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.NoRegionSelectedException*** – If no region has been selected to write to

write_command_to_files(*cmd_word_list*, *cmd_string*, *indent=False*, *outdent=False*, *no_instruction_number=False*)

Writes the binary command to the binary output file and, if the user has requested a text output for debug purposes, also write the text version to the text file. Setting the optional parameter 'indent' to True causes subsequent commands to be indented by two spaces relative to this one. Similarly, setting 'outdent' to True, reverses this spacing.

Parameters

- **cmd_word_list** (*bytearray*) – list of binary words to be added to the binary data specification file
- **cmd_string** (*str*) – string describing the command to be added to the report for the data specification file
- **indent** (*bool*) – if the following lines need to be indented
- **outdent** (*bool*) – if the following lines need to be out-dented
- **no_instruction_number** – if each report line should include also the address of the command in the file
- **no_instruction_number** – bool

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails

write_repeated_value(*data*, *repeats=1*, *repeats_is_register=False*, *data_type=<DataType.UINT32: 2>*)

Insert command to write a value one or more times to the current write pointer, causing the write pointer to move on by the number of bytes required to represent the data type. The data is passed as a parameter to this function

Parameters

- **data** (*float*) – the data to write as a float.
- **repeats** (*int*) –
  - If repeats_is_register is False, this parameter identifies the number of times to repeat the data, between 1 and 255 (default 1)
  - If repeats_is_register is True, this parameter identifies the register that contains the number of repeats.
- **repeats_is_register** (*bool*) – Indicates if the parameter repeats identifies the register containing the number of repeats of the value to write
- **data_type** (DataType) – the type to convert data to

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- *data_specification.exceptions.ParameterOutOfBoundsException* –
  - If repeats_register is None, and repeats is out of range
  - If repeats_register is not a valid register ID
  - If data_type is an integer type, and data has a fractional part
  - If data would overflow the data type
- *data_specification.exceptions.UnknownTypeException* – If the data type is not known
- *data_specification.exceptions.InvalidSizeException* – If the data size is invalid
- *data_specification.exceptions.NoRegionSelectedException* – If no region has been selected to write to

write_structure(*structure_id*, *repeats=1*, *repeats_is_register=False*)

Insert command to write a structure to the current write pointer, causing the current write pointer to move on by the number of bytes needed to represent the structure

Parameters

- **structure_id** (*int*) –
  - If called within a function, the ID of the structure to write, between 0 and 15
  - If called outside of a function, the ID of the structure argument, between 0 and 5
- **repeats** (*int*) –
  - If repeats_is_register is True, the ID of the register containing the number of repeats, between 0 and 15
  - If repeats_is_register is False, the number of repeats to write, between 0 and 255
- **repeats_is_register** (*bool*) – Identifies if repeats identifies a register

Returns No value returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- *data_specification.exceptions.ParameterOutOfBoundsException* –
  - If repeats_is_register is False and structure_id is not a valid ID
  - If repeats_is_register is True and structure_id
  - If the number of repeats is out of range

- ***data_specification.exceptions.NoRegionSelectedException*** – If no region has been selected to write to
- ***data_specification.exceptions.RegionExhaustedException*** – If the selected region has no more space

write_value(*data*, *data_type=<DataType.UINT32: 2>*)

Insert command to write a value one or more times to the current write pointer, causing the write pointer to move on by the number of bytes required to represent the data type. The data is passed as a parameter to this function

---

Note: This method used to have two extra parameters repeats and repeats_is_register. They have been removed here. If you need them, use write_repeated_value

---

Parameters

- **data** (*float*) – the data to write as a float.
- **data_type** (DataType) – the type to convert data to

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If repeats_register is None, and repeats is out of range
  - If repeats_register is not a valid register ID
  - If data_type is an integer type, and data has a fractional part
  - If data would overflow the data type
- ***data_specification.exceptions.UnknownTypeException*** – If the data type is not known
- ***data_specification.exceptions.InvalidSizeException*** – If the data size is invalid
- ***data_specification.exceptions.NoRegionSelectedException*** – If no region has been selected to write to

write_value_from_register(*data_register*, *repeats=1*, *repeats_is_register=False*, *data_type=<DataType.UINT32: 2>*)

Insert command to write a value one or more times at the write pointer of the current memory region, causing it to move. The data is contained in a register whose ID is passed to the function

Parameters

- **data_register** (*int*) – Identifies the register in which the data is stored.
- **repeats** (*int*) –

- If repeats_register is None, this parameter identifies the number of times to repeat the data, between 1 and 255 (default 1)
- If repeats_register is not None (i.e. has an integer value), the content of this parameter is disregarded

- **repeats_is_register** (*None or int*) – Identifies the register containing the number of repeats of the value to write
- **data_type** (DataType) – the type of the data held in the register

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **spinn_storage_handlers.exceptions.DataWriteException** – If a write to external storage fails
- ***data_specification.exceptions.ParameterOutOfBoundsException*** –
  - If repeats_register is None, and repeats is out of range
  - If repeats_register is not a valid register ID
  - If data_register is not a valid register ID
- ***data_specification.exceptions.UnknownTypeException*** – If the data type is not known
- ***data_specification.exceptions.NoRegionSelectedException*** – If no region has been selected to write to
- ***data_specification.exceptions.RegionExhaustedException*** – If the selected region has no more space

class data_specification.**MemoryRegion**(*unfilled*, *size*)

Bases: object

Memory region storage object

Parameters **unfilled** – if the region needs to be filled when written

Return type None

Raises **None** – this method does not raise any known exception

allocated_size

The size of the region

Returns the size of the region

Return type int

Raises **None** – this method does not raise any known exception

increment_write_pointer(*n_bytes*)

max_write_pointer

region_data

The container which holds the data written in this region

Returns the region data container

Return type bytearray

Raises **None** – this method does not raise any known exception

remaining_space

The space between the current write pointer and the end of the/ region

Returns the number of bytes in the region that can be written

Return type int

Raises **None** – this method does not raise any known exception

unfilled

Whether the region is marked as not fillable

Returns True if the region will not contain any data at write time

Return type bool

Raises **None** – this method does not raise any known exception

write_pointer

class data_specification.**MemoryRegionCollection**(*n_regions*)

Bases: object

Collection of memory regions.

Create a new MemoryRegionCollection with the given number of regions.

count_used_regions()

Return type int

is_empty(*region*)

Return type bool

is_unfilled(*region*)

Return type bool

needs_to_write_region(*region*)

Helper method which determines if a unfilled region actually needs to be written (optimisation to stop large data files).

Parameters **region** – the region ID to which the test is being ran on

Returns a boolean stating if the region needs to be written

Return type bool

Raises ***NoRegionSelectedException*** – when the ID is beyond the expected region range

regions

Return type iterable(*MemoryRegion*)

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

data_specification, **??**
data_specification.abstract_executor_functions, 14
data_specification.constants, 15
data_specification.data_specification_executor, 15
data_specification.data_specification_executor_functions, 17
data_specification.data_specification_generator, 19
data_specification.enums, 11
data_specification.enums.arithemetic_operation, 7
data_specification.enums.commands, 7
data_specification.enums.condition, 9
data_specification.enums.data_type, 9
data_specification.enums.logic_operation, 10
data_specification.enums.random_number_generator, 11
data_specification.exceptions, 44
data_specification.memory_region, 49
data_specification.memory_region_collection, 50
data_specification.utility_calls, 50

Index

A

AbstractExecutorFunctions (*class in data_specification.abstract_executor_functions*), 14
ADD (*data_specification.enums.arithemetic_operation.ArithmeticOperation attribute*), 7
ADD (*data_specification.enums.ArithmeticOperation attribute*), 11
ALIGN_WR_PTR (*data_specification.enums.Commands attribute*), 11
ALIGN_WR_PTR (*data_specification.enums.commands.Commands attribute*), 7
align_write_pointer() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 19
align_write_pointer() (*data_specification.DataSpecificationGenerator method*), 55
allocated_size (*data_specification.memory_region.MemoryRegion attribute*), 49
allocated_size (*data_specification.MemoryRegion attribute*), 80
AND (*data_specification.enums.logic_operation.LogicOperation attribute*), 10
AND (*data_specification.enums.LogicOperation attribute*), 13
ARITH_OP (*data_specification.enums.Commands attribute*), 11
ARITH_OP (*data_specification.enums.commands.Commands attribute*), 8
ArithmeticOperation (*class in data_specification.enums*), 11
ArithmeticOperation (*class in data_specification.enums.arithemetic_operation*), 7

B

BLOCK_COPY (*data_specification.enums.Commands attribute*), 11
BLOCK_COPY (*data_specification.enums.commands.Commands attribute*), 8
BREAK (*data_specification.enums.Commands attribute*), 11
BREAK (*data_specification.enums.commands.Commands attribute*), 8
BREAK_LOOP (*data_specification.enums.Commands attribute*), 11
BREAK_LOOP (*data_specification.enums.commands.Commands attribute*), 8
break_loop() (*data_specification.data_specification_generator.DataSp method*), 20
break_loop() (*data_specification.DataSpecificationGenerator method*), 56

C

call_arithmetic_operation() (*data_specification.data_specification_generator.DataSpecificati method*), 20
call_arithmetic_operation() (*data_specification.DataSpecificationGenerator method*), 56
call_function() (*data_specification.data_specification_generator.Da method*), 21
call_function() (*data_specification.DataSpecificationGenerator method*), 57
call_random_distribution() (*data_specification.data_specification_generator.DataSpecificati method*), 21
call_random_distribution() (*data_specification.DataSpecificationGenerator method*), 58
cmd_size (*data_specification.data_specification_executor_functions.Dat attribute*), 17
cmd_size (*data_specification.DataSpecificationExecutorFunctions attribute*), 53
Commands (*class in data_specification.enums*), 11
Commands (*class in data_specification.enums.commands*), 7

comment() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 22
comment() (*data_specification.DataSpecificationGenerator method*), 58
Condition (*class in data_specification.enums*), 12
Condition (*class in data_specification.enums.condition*), 9
conditionals (*data_specification.data_specification_generator.DataSpecificationGenerator attribute*), 22
conditionals (*data_specification.DataSpecificationGenerator attribute*), 58
CONSTRUCT (*data_specification.enums.Commands attribute*), 11
CONSTRUCT (*data_specification.enums.commands.Commands attribute*), 8
COPY_PARAM (*data_specification.enums.Commands attribute*), 11
COPY_PARAM (*data_specification.enums.commands.Commands attribute*), 8
COPY_STRUCT (*data_specification.enums.Commands attribute*), 11
COPY_STRUCT (*data_specification.enums.commands.Commands attribute*), 8
copy_structure() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 22
copy_structure() (*data_specification.DataSpecificationGenerator method*), 58
copy_structure_parameter() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 23
copy_structure_parameter() (*data_specification.DataSpecificationGenerator method*), 59
count_used_regions() (*data_specification.memory_region_collection.MemoryRegionCollection method*), 50
count_used_regions() (*data_specification.MemoryRegionCollection method*), 81
create_cmd() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 23
create_cmd() (*data_specification.DataSpecificationGenerator method*), 60
current_region (*data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions attribute*), 17
current_region (*data_specification.data_specification_generator.DataSpecificationGenerator attribute*), 24
current_region (*data_specification.DataSpecificationExecutorFunctions attribute*), 53
current_region (*data_specification.DataSpecificationGenerator attribute*), 60

D

data_len (*data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions attribute*), 17
data_len (*data_specification.DataSpecificationExecutorFunctions attribute*), 53
data_specification (*module*), 1, 51
data_specification.abstract_executor_functions (*module*), 14
data_specification.constants (*module*), 15
data_specification.data_specification_executor (*module*), 15
data_specification.data_specification_executor_functions (*module*), 17
data_specification.data_specification_generator (*module*), 19
data_specification.enums (*module*), 11
data_specification.enums.arithemetic_operation (*module*), 7
data_specification.enums.commands (*module*), 7
data_specification.enums.condition (*module*), 9
data_specification.enums.data_type (*module*), 9
data_specification.enums.logic_operation (*module*), 10
data_specification.enums.random_number_generator (*module*), 11
data_specification.exceptions (*module*), 44
data_specification.memory_region (*module*), 49
data_specification.memory_region_collection (*module*), 50
data_specification.utility_calls (*module*), 50
DataSpecificationException, 44
DataSpecificationExecutor (*class in data_specification*), 52
DataSpecificationExecutor (*class in data_specification.data_specification_executor*), 15
DataSpecificationExecutorFunctions (*class in data_specification*), 53
DataSpecificationExecutorFunctions (*class in data_specification.data_specification_executor_functions*), 17
DataSpecificationGenerator (*class in data_specification*), 55
DataSpecificationGenerator (*class in data_specification.data_specification_generator*), 19
DataSpecificationSyntaxError, 44
DataType (*class in data_specification.enums*), 12
DataType (*class in data_specification.enums.data_type*), 9
DataUndefinedWriteError, 44

DECLARE_RANDOM_DIST (*data_specification.enums.Commands attribute*), 11
DECLARE_RANDOM_DIST (*data_specification.enums.commands.Commands attribute*), 8
declare_random_number_generator() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 24
declare_random_number_generator() (*data_specification.DataSpecificationGenerator method*), 60
DECLARE_RNG (*data_specification.enums.Commands attribute*), 11
DECLARE_RNG (*data_specification.enums.commands.Commands attribute*), 8
declare_uniform_random_distribution() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 25
declare_uniform_random_distribution() (*data_specification.DataSpecificationGenerator method*), 61
define_break() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 25
define_break() (*data_specification.DataSpecificationGenerator method*), 62
define_structure() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 26
define_structure() (*data_specification.DataSpecificationGenerator method*), 62
dest_reg (*data_specification.data_specification_executor.DataSpecificationExecutor attribute*), 17
dest_reg (*data_specification.DataSpecificationExecutor attribute*), 53
dsef (*data_specification.data_specification_executor.DataSpecificationExecutor attribute*), 16
dsef (*data_specification.DataSpecificationExecutor attribute*), 52
DuplicateParameterException, 44

E

ELSE (*data_specification.enums.Commands attribute*), 11
ELSE (*data_specification.enums.commands.Commands attribute*), 8
else_conditional() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 26
else_conditional() (*data_specification.DataSpecificationGenerator method*), 62
encode() (*data_specification.enums.data_type.DataType method*), 10
encode() (*data_specification.enums.DataType method*), 13
encode_as_int() (*data_specification.enums.data_type.DataType method*), 10
encode_as_int() (*data_specification.enums.DataType method*), 13
end_conditional() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 26
end_conditional() (*data_specification.DataSpecificationGenerator method*), 63
END_CONSTRUCTOR (*data_specification.enums.Commands attribute*), 11
END_CONSTRUCTOR (*data_specification.enums.commands.Commands attribute*), 8
end_function() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 27
end_function() (*data_specification.DataSpecificationGenerator method*), 63
END_IF (*data_specification.enums.Commands attribute*), 11
END_IF (*data_specification.enums.commands.Commands attribute*), 8
END_LOOP (*data_specification.enums.Commands attribute*), 11
END_LOOP (*data_specification.enums.commands.Commands attribute*), 8
end_loop() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 27
end_loop() (*data_specification.DataSpecificationGenerator method*), 63
END_SPEC (*data_specification.enums.Commands attribute*), 11
END_SPEC (*data_specification.enums.commands.Commands attribute*), 8
end_specification() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 27
end_specification() (*data_specification.DataSpecificationGenerator method*), 63
END_STRUCT (*data_specification.enums.Commands attribute*), 11
END_STRUCT (*data_specification.enums.commands.Commands attribute*), 8
EQUAL (*data_specification.enums.Condition attribute*), 12
EQUAL (*data_specification.enums.condition.Condition attribute*), 9
execute() (*data_specification.data_specification_executor.DataSpecificationExecutor method*), 16

execute() (data_specification.DataSpecificationExecutor method), 52
execute_align_wr_ptr() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 14
execute_arith_op() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 14
execute_block_copy() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 14
execute_break() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 14
execute_break() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 17
execute_break() (data_specification.DataSpecificationExecutorFunctions method), 53
execute_break_loop() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 14
execute_construct() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 14
execute_copy_param() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 14
execute_copy_struct() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 14
execute_declare_rng() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 14
execute_else() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 14
execute_end_constructor() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 14
execute_end_if() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 14
execute_end_loop() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 14
execute_end_spec() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 14
execute_end_spec() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 17
execute_end_spec() (data_specification.DataSpecificationExecutorFunctions method), 54
execute_end_struct() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 14
execute_free() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 14
execute_get_random_rumber() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 14
execute_get_wr_ptr() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 14
execute_if() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 14
execute_logic_op() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 14
execute_loop() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 15
execute_mv() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 15
execute_mv() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 17
execute_mv() (data_specification.DataSpecificationExecutorFunctions method), 54
execute_ptr() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 15
execute_print_struct() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 15
execute_print_txt() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 15
execute_print_val() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 15
execute_reserve() (data_specification.abstract_executor_functions.AbstractExecutorFunctions method), 15
execute_reserve() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 17
execute_reserve() (data_specification.DataSpecificationExecutorFunctions method), 54
execute_set_ptr() (data_specification.abstract_executor_functions.AbstractExecutorFunctions

method), 15
execute_set_wr_ptr() (*data_specification.abstract_executor_functions.AbstractExecutorFunctions method*), 15
execute_set_wr_ptr() (*data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method*), 18
execute_set_wr_ptr() (*data_specification.DataSpecificationExecutorFunctions method*), 54
execute_start_constructor() (*data_specification.abstract_executor_functions.AbstractExecutorFunctions method*), 15
execute_start_struct() (*data_specification.abstract_executor_functions.AbstractExecutorFunctions method*), 15
execute_struct_elem() (*data_specification.abstract_executor_functions.AbstractExecutorFunctions method*), 15
execute_switch_focus() (*data_specification.abstract_executor_functions.AbstractExecutorFunctions method*), 15
execute_switch_focus() (*data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method*), 18
execute_switch_focus() (*data_specification.DataSpecificationExecutorFunctions method*), 54
execute_write() (*data_specification.abstract_executor_functions.AbstractExecutorFunctions method*), 15
execute_write() (*data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method*), 18
execute_write() (*data_specification.DataSpecificationExecutorFunctions method*), 54
execute_write_array() (*data_specification.abstract_executor_functions.AbstractExecutorFunctions method*), 15
execute_write_array() (*data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method*), 18
execute_write_array() (*data_specification.DataSpecificationExecutorFunctions method*), 55
execute_write_param() (*data_specification.abstract_executor_functions.AbstractExecutorFunctions method*), 15
execute_write_param_component() (*data_specification.abstract_executor_functions.AbstractExecutorFunctions method*), 15
execute_write_struct() (*data_specification.abstract_executor_functions.AbstractExecutorFunctions method*), 15
ExecuteBreakInstruction, 44

F

FLOAT_32 (*data_specification.enums.data_type.DataType attribute*), 9
FLOAT_32 (*data_specification.enums.DataType attribute*), 12
FLOAT_64 (*data_specification.enums.data_type.DataType attribute*), 9
FLOAT_64 (*data_specification.enums.DataType attribute*), 13
FREE (*data_specification.enums.Commands attribute*), 11
FREE (*data_specification.enums.commands.Commands attribute*), 8
free_memory_region() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 27
free_memory_region() (*data_specification.DataSpecificationGenerator method*), 64
function_slots (*data_specification.data_specification_generator.DataSpecificationGenerator attribute*), 28
function_slots (*data_specification.DataSpecificationGenerator attribute*), 64
FunctionInUseException, 44

G

get_constructed_data_size() (*data_specification.data_specification_executor.DataSpecificationExecutor method*), 16
get_constructed_data_size() (*data_specification.DataSpecificationExecutor method*), 53
get_data_spec_and_file_writer_filename() (*in module data_specification.utility_calls*), 50
get_header() (*data_specification.data_specification_executor.DataSpecificationExecutor method*), 16
get_header() (*data_specification.DataSpecificationExecutor method*), 53
get_pointer_table() (*data_specification.data_specification_executor.DataSpecificationExecutor method*), 16
get_pointer_table() (*data_specification.DataSpecificationExecutor method*), 53
GET_RANDOM_NUMBER (*data_specification.enums.Commands attribute*), 11
GET_RANDOM_NUMBER (*data_specification.enums.commands.Commands attribute*), 8
get_region() (*data_specification.data_specification_executor.DataSpecificationExecutor method*), 16
get_region() (*data_specification.DataSpecificationExecutor method*), 53

get_region_base_address_offset() (*in module data_specification.utility_calls*), 51
get_report_writer() (*in module data_specification.utility_calls*), 51
get_structure_value() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 28
get_structure_value() (*data_specification.DataSpecificationGenerator method*), 64
GET_WR_PTR (*data_specification.enums.Commands attribute*), 11
GET_WR_PTR (*data_specification.enums.commands.Commands attribute*), 8
GREATER_THAN (*data_specification.enums.Condition attribute*), 12
GREATER_THAN (*data_specification.enums.condition.Condition attribute*), 9
GREATER_THAN_OR_EQUAL (*data_specification.enums.Condition attribute*), 12
GREATER_THAN_OR_EQUAL (*data_specification.enums.condition.Condition attribute*), 9

I

IF (*data_specification.enums.Commands attribute*), 11
IF (*data_specification.enums.commands.Commands attribute*), 8
increment_write_pointer() (*data_specification.memory_region.MemoryRegion method*), 49
increment_write_pointer() (*data_specification.MemoryRegion method*), 80
instruction_counter (*data_specification.data_specification_generator.DataSpecificationGenerator attribute*), 28
instruction_counter (*data_specification.DataSpecificationGenerator attribute*), 65
INT16 (*data_specification.enums.data_type.DataType attribute*), 9
INT16 (*data_specification.enums.DataType attribute*), 13
INT32 (*data_specification.enums.data_type.DataType attribute*), 9
INT32 (*data_specification.enums.DataType attribute*), 13
INT64 (*data_specification.enums.data_type.DataType attribute*), 9
INT64 (*data_specification.enums.DataType attribute*), 13
INT8 (*data_specification.enums.data_type.DataType attribute*), 9
INT8 (*data_specification.enums.DataType attribute*), 13
InvalidCommandException, 44
InvalidOperationException, 45
InvalidParameterException, 45
is_empty() (*data_specification.memory_region_collection.MemoryRegionCollection method*), 50
is_empty() (*data_specification.MemoryRegionCollection method*), 81
is_unfilled() (*data_specification.memory_region_collection.MemoryRegionCollection method*), 50
is_unfilled() (*data_specification.MemoryRegionCollection method*), 81

L

LEFT_SHIFT (*data_specification.enums.logic_operation.LogicOperation attribute*), 10
LEFT_SHIFT (*data_specification.enums.LogicOperation attribute*), 14
LESS_THAN (*data_specification.enums.Condition attribute*), 12
LESS_THAN (*data_specification.enums.condition.Condition attribute*), 9
LESS_THAN_OR_EQUAL (*data_specification.enums.Condition attribute*), 12
LESS_THAN_OR_EQUAL (*data_specification.enums.condition.Condition attribute*), 9
LOGIC_OP (*data_specification.enums.Commands attribute*), 11
LOGIC_OP (*data_specification.enums.commands.Commands attribute*), 8
logical_and() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 28
logical_and() (*data_specification.DataSpecificationGenerator method*), 65
logical_left_shift() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 29
logical_left_shift() (*data_specification.DataSpecificationGenerator method*), 65
logical_not() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 30
logical_not() (*data_specification.DataSpecificationGenerator method*), 66
logical_or() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 30
logical_or() (*data_specification.DataSpecificationGenerator method*), 66
logical_right_shift() (*data_specification.data_specification_generator.DataSpecificationGenerator*

method), 31
logical_right_shift() (*data_specification.DataSpecificationGenerator method*), 67
logical_xor() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 31
logical_xor() (*data_specification.DataSpecificationGenerator method*), 68
LogicOperation (*class in data_specification.enums*), 13
LogicOperation (*class in data_specification.enums.logic_operation*), 10
LOOP (*data_specification.enums.Commands attribute*), 12
LOOP (*data_specification.enums.commands.Commands attribute*), 8

M

MAGIC_NUMBER (*data_specification.data_specification_generator.DataSpecificationGenerator attribute*), 19
MAGIC_NUMBER (*data_specification.DataSpecificationGenerator attribute*), 55
max (*data_specification.enums.data_type.DataType attribute*), 10
max (*data_specification.enums.DataType attribute*), 13
max_write_pointer (*data_specification.memory_region.MemoryRegion attribute*), 49
max_write_pointer (*data_specification.MemoryRegion attribute*), 80
mem_regions (*data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions attribute*), 18
mem_regions (*data_specification.DataSpecificationExecutorFunctions attribute*), 55
mem_slots (*data_specification.data_specification_generator.DataSpecificationGenerator attribute*), 32
mem_slots (*data_specification.DataSpecificationGenerator attribute*), 68
memory_space (*data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions attribute*), 18
memory_space (*data_specification.DataSpecificationExecutorFunctions attribute*), 55
MemoryRegion (*class in data_specification*), 80
MemoryRegion (*class in data_specification.memory_region*), 49
MemoryRegionCollection (*class in data_specification*), 81
MemoryRegionCollection (*class in data_specification.memory_region_collection*), 50
MERSENNE_TWISTER (*data_specification.enums.random_number_generator.RandomNumberGenerator attribute*), 11
MERSENNE_TWISTER (*data_specification.enums.RandomNumberGenerator attribute*), 14
min (*data_specification.enums.data_type.DataType attribute*), 10
min (*data_specification.enums.DataType attribute*), 13
MULTIPLY (*data_specification.enums.arithemetic_operation.ArithmeticOperation attribute*), 7
MULTIPLY (*data_specification.enums.ArithmeticOperation attribute*), 11
MV (*data_specification.enums.Commands attribute*), 12
MV (*data_specification.enums.commands.Commands attribute*), 8

N

needs_to_write_region() (*data_specification.memory_region_collection.MemoryRegionCollection method*), 50
needs_to_write_region() (*data_specification.MemoryRegionCollection method*), 81
NestedFunctionException, 45
no_operation() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 32
no_operation() (*data_specification.DataSpecificationGenerator method*), 68
NoMoreException, 45
NOP (*data_specification.enums.Commands attribute*), 12
NOP (*data_specification.enums.commands.Commands attribute*), 8
NoRegionSelectedException, 45
NOT (*data_specification.enums.logic_operation.LogicOperation attribute*), 10
NOT (*data_specification.enums.LogicOperation attribute*), 14
NOT_EQUALS (*data_specification.enums.Condition attribute*), 12
NOT_EQUALS (*data_specification.enums.condition.Condition attribute*), 9
NotAllocatedException, 45
numpy_typename (*data_specification.enums.data_type.DataType attribute*), 10
numpy_typename (*data_specification.enums.DataType attribute*), 13

O

ongoing_function_definition (*data_specification.data_specification_generator.DataSpecificationGenerator attribute*), 32
ongoing_function_definition (*data_specification.DataSpecificationGenerator attribute*), 68
ongoing_loop (*data_specification.data_specification_generator.DataSpecificationGenerator attribute*), 32

ongoing_loop (*data_specification.DataSpecificationGenerator attribute*), 68
opcode (*data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions attribute*), 18
opcode (*data_specification.DataSpecificationExecutorFunctions attribute*), 55
OR (*data_specification.enums.logic_operation.LogicOperation attribute*), 10
OR (*data_specification.enums.LogicOperation attribute*), 14

P

ParameterOutOfBoundsException, 46
PRINT_STRUCT (*data_specification.enums.Commands attribute*), 12
PRINT_STRUCT (*data_specification.enums.commands.Commands attribute*), 8
print_struct() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 32
print_struct() (*data_specification.DataSpecificationGenerator method*), 68
print_text() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 33
print_text() (*data_specification.DataSpecificationGenerator method*), 69
PRINT_TXT (*data_specification.enums.Commands attribute*), 12
PRINT_TXT (*data_specification.enums.commands.Commands attribute*), 8
PRINT_VAL (*data_specification.enums.Commands attribute*), 12
PRINT_VAL (*data_specification.enums.commands.Commands attribute*), 8
print_value() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 33
print_value() (*data_specification.DataSpecificationGenerator method*), 69

R

random_distribution (*data_specification.data_specification_generator.DataSpecificationGenerator attribute*), 34
random_distribution (*data_specification.DataSpecificationGenerator attribute*), 70
RandomNumberDistributionInUseException, 46
RandomNumberGenerator (*class in data_specification.enums*), 14
RandomNumberGenerator (*class in data_specification.enums.random_number_generator*), 11
READ (*data_specification.enums.Commands attribute*), 12
READ (*data_specification.enums.commands.Commands attribute*), 8
READ_PARAM (*data_specification.enums.Commands attribute*), 12
READ_PARAM (*data_specification.enums.commands.Commands attribute*), 8
read_value() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 34
read_value() (*data_specification.DataSpecificationGenerator method*), 70
REFORMAT (*data_specification.enums.Commands attribute*), 12
REFORMAT (*data_specification.enums.commands.Commands attribute*), 8
region_data (*data_specification.memory_region.MemoryRegion attribute*), 49
region_data (*data_specification.MemoryRegion attribute*), 80
region_sizes (*data_specification.data_specification_generator.DataSpecificationGenerator attribute*), 34
region_sizes (*data_specification.DataSpecificationGenerator attribute*), 70
RegionExhaustedException, 46
RegionInUseException, 46
RegionNotAllocatedException, 47
RegionOutOfBoundsException, 47
regions (*data_specification.memory_region_collection.MemoryRegionCollection attribute*), 50
regions (*data_specification.MemoryRegionCollection attribute*), 81
RegionUnfilledException, 47
registers (*data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions attribute*), 18
registers (*data_specification.DataSpecificationExecutorFunctions attribute*), 55
remaining_space (*data_specification.memory_region.MemoryRegion attribute*), 49
remaining_space (*data_specification.MemoryRegion attribute*), 81
report_writer (*data_specification.data_specification_generator.DataSpecificationGenerator attribute*), 34
report_writer (*data_specification.DataSpecificationGenerator attribute*), 70
RESERVE (*data_specification.enums.Commands attribute*), 12
RESERVE (*data_specification.enums.commands.Commands attribute*), 8
reserve_memory_region() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 34
reserve_memory_region() (*data_specification.DataSpecificationGenerator method*), 70
RIGHT_SHIFT (*data_specification.enums.logic_operation.LogicOperation*

attribute), 10
RIGHT_SHIFT (*data_specification.enums.LogicOperation attribute*), 14
rng (*data_specification.data_specification_generator.DataSpecificationGenerator attribute*), 34
rng (*data_specification.DataSpecificationGenerator attribute*), 71
RNGInUseException, 46

S

S015 (*data_specification.enums.data_type.DataType attribute*), 9
S015 (*data_specification.enums.DataType attribute*), 13
S031 (*data_specification.enums.data_type.DataType attribute*), 9
S031 (*data_specification.enums.DataType attribute*), 13
S063 (*data_specification.enums.data_type.DataType attribute*), 9
S063 (*data_specification.enums.DataType attribute*), 13
S07 (*data_specification.enums.data_type.DataType attribute*), 9
S07 (*data_specification.enums.DataType attribute*), 13
S1615 (*data_specification.enums.data_type.DataType attribute*), 9
S1615 (*data_specification.enums.DataType attribute*), 13
S3231 (*data_specification.enums.data_type.DataType attribute*), 9
S3231 (*data_specification.enums.DataType attribute*), 13
S87 (*data_specification.enums.data_type.DataType attribute*), 10
S87 (*data_specification.enums.DataType attribute*), 13
save_write_pointer() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 34
save_write_pointer() (*data_specification.DataSpecificationGenerator method*), 71
scale (*data_specification.enums.data_type.DataType attribute*), 10
scale (*data_specification.enums.DataType attribute*), 13
set_register_value() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 35
set_register_value() (*data_specification.DataSpecificationGenerator method*), 71
set_structure_value() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 35
set_structure_value() (*data_specification.DataSpecificationGenerator method*), 72
SET_WR_PTR (*data_specification.enums.Commands attribute*), 12
SET_WR_PTR (*data_specification.enums.commands.Commands attribute*), 8
set_write_pointer() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 36
set_write_pointer() (*data_specification.DataSpecificationGenerator method*), 73
size (*data_specification.enums.data_type.DataType attribute*), 10
size (*data_specification.enums.DataType attribute*), 13
space_allocated (*data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions attribute*), 18
space_allocated (*data_specification.DataSpecificationExecutorFunctions attribute*), 55
spec_reader (*data_specification.data_specification_executor.DataSpecificationExecutor attribute*), 16
spec_reader (*data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions attribute*), 18
spec_reader (*data_specification.DataSpecificationExecutor attribute*), 53
spec_reader (*data_specification.DataSpecificationExecutorFunctions attribute*), 55
spec_writer (*data_specification.data_specification_generator.DataSpecificationGenerator attribute*), 37
spec_writer (*data_specification.DataSpecificationGenerator attribute*), 73
src1_reg (*data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions attribute*), 18
src1_reg (*data_specification.DataSpecificationExecutorFunctions attribute*), 55
src2_reg (*data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions attribute*), 18
src2_reg (*data_specification.DataSpecificationExecutorFunctions attribute*), 55
start_conditional() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 37
start_conditional() (*data_specification.DataSpecificationGenerator method*), 73
START_CONSTRUCTOR (*data_specification.enums.Commands attribute*), 12
START_CONSTRUCTOR (*data_specification.enums.commands.Commands attribute*), 8
start_function() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 37
start_function() (*data_specification.DataSpecificationGenerator method*), 74

start_loop() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 38
start_loop() (*data_specification.DataSpecificationGenerator method*), 74
START_STRUCT (*data_specification.enums.Commands attribute*), 12
START_STRUCT (*data_specification.enums.commands.Commands attribute*), 8
STRUCT_ELEM (*data_specification.enums.Commands attribute*), 12
STRUCT_ELEM (*data_specification.enums.commands.Commands attribute*), 8
struct_encoding (*data_specification.enums.data_type.DataType attribute*), 10
struct_encoding (*data_specification.enums.DataType attribute*), 13
struct_slots (*data_specification.data_specification_generator.DataSpecificationGenerator attribute*), 39
struct_slots (*data_specification.DataSpecificationGenerator attribute*), 75
StructureInUseException, 47
SUBTRACT (*data_specification.enums.arithemetic_operation.ArithmeticOperation attribute*), 7
SUBTRACT (*data_specification.enums.ArithmeticOperation attribute*), 11
SWITCH_FOCUS (*data_specification.enums.Commands attribute*), 12
SWITCH_FOCUS (*data_specification.enums.commands.Commands attribute*), 8
switch_write_focus() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 39
switch_write_focus() (*data_specification.DataSpecificationGenerator method*), 75

T

TablePointerOutOfMemoryException, 47
txt_indent (*data_specification.data_specification_generator.DataSpecificationGenerator attribute*), 39
txt_indent (*data_specification.DataSpecificationGenerator attribute*), 76
TypeMismatchException, 48

U

U016 (*data_specification.enums.data_type.DataType attribute*), 10
U016 (*data_specification.enums.DataType attribute*), 13
U032 (*data_specification.enums.data_type.DataType attribute*), 10
U032 (*data_specification.enums.DataType attribute*), 13
U064 (*data_specification.enums.data_type.DataType attribute*), 10
U064 (*data_specification.enums.DataType attribute*), 13
U08 (*data_specification.enums.data_type.DataType attribute*), 10
U08 (*data_specification.enums.DataType attribute*), 13
U1616 (*data_specification.enums.data_type.DataType attribute*), 10
U1616 (*data_specification.enums.DataType attribute*), 13
U3232 (*data_specification.enums.data_type.DataType attribute*), 10
U3232 (*data_specification.enums.DataType attribute*), 13
U88 (*data_specification.enums.data_type.DataType attribute*), 10
U88 (*data_specification.enums.DataType attribute*), 13
UINT16 (*data_specification.enums.data_type.DataType attribute*), 10
UINT16 (*data_specification.enums.DataType attribute*), 13
UINT32 (*data_specification.enums.data_type.DataType attribute*), 10
UINT32 (*data_specification.enums.DataType attribute*), 13
UINT64 (*data_specification.enums.data_type.DataType attribute*), 10
UINT64 (*data_specification.enums.DataType attribute*), 13
UINT8 (*data_specification.enums.data_type.DataType attribute*), 10
UINT8 (*data_specification.enums.DataType attribute*), 13
UnfilledMemoryRegion (*data_specification.memory_region.MemoryRegion attribute*), 49
unfilled (*data_specification.MemoryRegion attribute*), 81
UnimplementedDSECommandError, 48
UnimplementedDSGCommandError, 48
UnknownConditionException, 48
UnknownTypeException, 48
UnknownVarLengthException, 48

V

VERSION (*data_specification.data_specification_generator.DataSpecification attribute*), 19
VERSION (*data_specification.DataSpecificationGenerator attribute*), 55

W

WRITE (*data_specification.enums.Commands attribute*), 12
WRITE (*data_specification.enums.commands.Commands attribute*), 8
WRITE_ARRAY (*data_specification.enums.Commands attribute*), 12

WRITE_ARRAY (*data_specification.enums.commands.Commands attribute*), 9
write_array() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 39
write_array() (*data_specification.DataSpecificationGenerator method*), 76
write_cmd() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 40
write_cmd() (*data_specification.DataSpecificationGenerator method*), 76
write_command_to_files() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 40
write_command_to_files() (*data_specification.DataSpecificationGenerator method*), 76
WRITE_PARAM (*data_specification.enums.Commands attribute*), 12
WRITE_PARAM (*data_specification.enums.commands.Commands attribute*), 9
WRITE_PARAM_COMPONENT (*data_specification.enums.Commands attribute*), 12
WRITE_PARAM_COMPONENT (*data_specification.enums.commands.Commands attribute*), 9
write_pointer (*data_specification.memory_region.MemoryRegion attribute*), 50
write_pointer (*data_specification.MemoryRegion attribute*), 81
write_repeated_value() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 41
write_repeated_value() (*data_specification.DataSpecificationGenerator method*), 77
WRITE_STRUCT (*data_specification.enums.Commands attribute*), 12
WRITE_STRUCT (*data_specification.enums.commands.Commands attribute*), 9
write_structure() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 41
write_structure() (*data_specification.DataSpecificationGenerator method*), 78
write_value() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 42
write_value() (*data_specification.DataSpecificationGenerator method*), 79
write_value_from_register() (*data_specification.data_specification_generator.DataSpecificationGenerator method*), 43
write_value_from_register() (*data_specification.DataSpecificationGenerator method*), 79
WriteToReadOnlyException, 48

X

XOR (*data_specification.enums.logic_operation.LogicOperation attribute*), 10
XOR (*data_specification.enums.LogicOperation attribute*), 14