DataSpecification Documentation

Release 6.0.0

Apr 08, 2021

Contents

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

This code depends on SpiNNUtils and SpiNNMachine (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 `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 `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

Main API

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

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

Parameters

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

Raises **IOError** – If a read or write fails

dsef

The executor functions themselves.

execute()

Executes the specification. This will result in a configuration of memory regions being done.

Raises

- **IOError** – If a read or write fails
- ***DataSpecificationException*** – If there is an error when executing the specification
- ***TablePointerOutOfMemoryException*** – 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.

Return type numpy.ndarray

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

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

Return type numpy.ndarray

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* or None

mem_regions
An enumeration of the mapping from region ID to region holder.

Return type iterable(tuple(int, *MemoryRegion* or None))

class data_specification.**DataSpecificationExecutorFunctions**(*spec_reader*, *memory_space*)
This class includes the function related to each of the commands of the data specification file.

Note: DSG operations not mentioned in this class will cause an error during DSE if used.

Parameters

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

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

Implements *BREAK*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.
- ***ExecuteBreakInstruction*** – Raises the exception to break the execution of the DSE

execute_end_spec(*cmd*)
This command marks the end of the specification program.

Implements *END_SPEC*

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

Returns A special marker to signal the end.

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_mv(*cmd*)

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

Implements *MV*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_reserve(*cmd*)

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

Implements *RESERVE*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.
- ***ParameterOutOfBoundsException*** – If the requested size of the region is beyond the available memory space

execute_set_wr_ptr(*cmd*)

This command sets the current write pointer.

Implements *SET_WR_PTR*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.
- ***NoRegionSelectedException*** – If there is no memory region selected for the set-ptr operation

execute_switch_focus(*cmd*)

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

Implements *SWITCH_FOCUS*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.
- ***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.

Implements *WRITE*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.
- ***NoRegionSelectedException*** – If there is no memory region selected for the write operation
- ***RegionNotAllocatedException*** – If the selected region has not been allocated memory space
- ***NoMoreException*** – If the selected region has not enough available memory to store the required data
- ***UnknownTypeLengthException*** – If the data type size is not 1, 2, 4, or 8 bytes

execute_write_array(*cmd*)

This command writes an array of values in the specified region.

Implements *WRITE_ARRAY*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.
- ***NoRegionSelectedException*** – If there is no memory region selected for the write operation
- ***RegionNotAllocatedException*** – If the selected region has not been allocated memory space
- ***NoMoreException*** – If the selected region has not enough available memory to store the required data

mem_regions

The collection of memory regions that can be written to.

Return type *MemoryRegionCollection*

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

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

Parameters

- **spec_writer** (*RawIOBase*) – The object to write the specification to
- **report_writer** (*TextIOBase or None*) – 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 **IOError** – If a write to external storage fails

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 $2^{\text{log\_block\_size}}$), 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 or None*) – The ID of a register where the write pointer will be written to once it has been updated, between 0 and 15, or None if no such writing is to be done

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***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
- ***RegionOutOfBoundsException*** – If the move of the pointer would put it outside of the current region
- ***NoRegionSelectedException*** – If no region has been selected

break_loop()

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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***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

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If `register_id` is not a valid register ID
  - 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
- ***InvalidOperationException*** – 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 (int)*) – A list of structure IDs that will be passed into the function, each between 0 and 15

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If the function ID is not valid
  - If any of the structure IDs are not valid
- ***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`
- ***WrongParameterNumberException*** – If a function is called with a wrong number of parameters

- **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

Raises

- **DataUndefinedWriterException** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- **NotAllocatedException** – If the random distribution ID was not previously declared
- **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

Raises

- **DataUndefinedWriterException** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails

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

Raises

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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***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
- ***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` (binary data 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 tuple(bytearray, str)

Raises

- ***ParameterOutOfBoundsException*** –
  - If `data_type` is an integer type, and `data` has a fractional part
  - If `data` would overflow the data type
- ***UnknownTypeException*** – If the data type is not known
- ***InvalidSizeException*** – If the data size is invalid

current_region

The ID of the current DSG region we're writing to. If not yet writing to any region, `None`.

Return type int or None

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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***UnknownTypeException*** – If the `rng_type` is not one of the allowed values
- ***ParameterOutOfBoundsException*** –
  - If the `seed` is too big or too small
  - If the `rng_id` is not in the allowed range
- ***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** (*int*) – ID of the distribution being set up
- **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

Raises

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

define_break()

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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – 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(tuple(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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails

- *NoMoreException* – If there are no more spaces for new structures
- *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`
- *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.

Raises

- *DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *IOError* – If a write to external storage fails
- *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

Raises

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

end_function()

Insert command to mark the end of a function definition

Raises *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.

Raises

- *DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *IOError* – If a write to external storage fails
- *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)

Raises

- **DataUndefinedWriterException** – If the binary specification file writer has not been initialised
- **IOError** – 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

Raises

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

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 `parameter_index` is a register ID containing the ID of the parameter/element to copy

Raises

- **DataUndefinedWriterException** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- **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
- **NotAllocatedException** – If the structure requested has not been declared

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.

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

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If `register_id` is not a valid register ID
  - 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.

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

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If `register_id` is not a valid register ID

- 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 not 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` is a register ID

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If `register_id` is not a valid register ID
  - If `operand_is_register` is True and `operand` 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.

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

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –

- If `register_id` is not a valid register ID
- 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.

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

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If `register_id` is not a valid register ID
  - 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.

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

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If `register_id` is not a valid register ID
  - 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

no_operation()

Insert command to execute nothing

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails

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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***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

- **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** (*str*) – The character encoding to use for the string. Defaults to ASCII.

Raises

- **DataUndefinedWriterException** – If the binary specification file writer has not been initialised
- **IOError** – 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 or int*) –
  - 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 value of type given by `data_type`
- **value_is_register** (*bool*) – Indicates if `value` is a register
- **data_type** (`DataType`) – The type of the data to be printed

Raises

- **DataUndefinedWriterException** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- **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
- **UnknownTypeException** – If `data_type` is not a valid data type

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.

Raises **ParameterOutOfBoundsException** – If `dest_id` is out of range for a register ID

region_sizes
A list of sizes of each region that has been reserved.

Note: The list will include `0` for each non-reserved region.

Return type list(int)

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 or None*) – 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

Raises

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

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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** – If the `register_id` is not a valid register ID
- ***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** (*int or 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 is a value of the type given by `data_type`

- **data_is_register** (*bool*) – Indicates if `data` is a register ID
- **data_type** (`DataType`) – The type of the data to be assigned

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***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
- ***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 or int
  - 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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***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 `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

- ***NotAllocatedException*** – If the structure requested has not been declared
- ***UnknownTypeException*** – If the data type is unknown

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 `address` is a register ID
- **relative_to_current** (*bool*) – Indicates if `address` (or the value read from that register when `address_is_register` is True) is to be added to the current address, or used as an absolute address from the start of the current region

Raises

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

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

Raises

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

- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If `register_id` is not a valid register ID
  - if `value_is_register` is True and `value` is not a valid register ID
- ***UnknownConditionException*** – If `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(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.

Raises

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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If `counter_register_id` is not a valid register ID
  - If `start_is_register` is True and `start` is not a valid register ID
  - If `end_is_register` is True and `end` 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

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

Raises

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

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(int) or list(float) or ndarray*) – An array of words to be written
- **data_type** (`DataType`) – Type of data contained in the array

Raises

- **DataUndefinedWriterException** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- **NoRegionSelectedException** – If no region has been 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

Raises

- **IOError** – If a write to external storage fails
- **NoRegionSelectedException** – If no region has been selected

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 or int*) – 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

Raises

- **DataUndefinedWriterException** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- **ParameterOutOfBoundsException** –
  - If `repeats_is_register` is False, and `repeats` is out of range
  - If `repeats_is_register` is True, and `repeats` 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

- ***UnknownTypeException*** – If the data type is not known
- ***InvalidSizeException*** – If the data size is invalid
- ***NoRegionSelectedException*** – If no region has been selected

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*) – Whether `repeats` identifies a register

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If `structure_id` is not a valid structure ID
  - If `repeats_is_register` is False and `repeats` is not in range
  - If `repeats_is_register` is True and `repeats` is not a valid register ID
- ***NoRegionSelectedException*** – If no region has been selected
- ***RegionExhaustedException*** – If the selected region has no more space

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

Insert command to write a value (once) 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** (*int or float*) – the data to write as a float.
- **data_type** (`DataType`) – the type to convert `data` to

Raises

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

- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If `data_type` is an integer type, and `data` has a fractional part
  - If `data` would overflow the data type
- ***UnknownTypeException*** – If the data type is not known
- ***InvalidSizeException*** – If the data size is invalid
- ***NoRegionSelectedException*** – If no region has been selected

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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If `repeats_is_register` is False, and `repeats` is out of range
  - If `repeats_is_register` is True, and `repeats` is not a valid register ID
  - If `data_register` is not a valid register ID
- ***UnknownTypeException*** – If the data type is not known
- ***NoRegionSelectedException*** – If no region has been selected
- ***RegionExhaustedException*** – If the selected region has no more space

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

Memory region storage object.

Parameters

- **unfilled** (*bool*) – if the region should not be written to by the data specification (i.e., because the vertex uses it as a working data region or an output region)
- **size** (*int*) – the size of the region, in bytes

allocated_size

The size of the region.

Return type int

increment_write_pointer(*n_bytes*)
Advance the write pointer.

Parameters **n_bytes** (*int*) – The number of bytes to advance the pointer by.

max_write_pointer
The max point where if written over, it will cause an error.

Return type int

region_data
The buffer which holds the data written in this region.

Return type bytearray

remaining_space
The space between the current write pointer and the end of the region, which is the number of bytes remaining in the region that can be written.

Return type int

unfilled
Whether the region is marked as not fillable; unfilled regions will not contain any data at write time.

Return type bool

write_pointer
The position in the buffer where data will be written to next.

Return type int

class data_specification.**MemoryRegionCollection**(*n_regions*)
Collection of memory regions.

Parameters **n_regions** (*int*) – The number of regions in the collection.

count_used_regions()
The number of regions in the collection that are used.

Return type int

is_empty(*region*)

Parameters **region** (*int*) – The ID of the region

Return type bool

is_unfilled(*region*)

Parameters **region** (*int*) – The ID of the 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** (*int*) – the region ID to which the test is being ran on

Returns whether the region needs to be written

Return type bool

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

regions

The regions in the collection.

Return type iterable(*MemoryRegion*)

CHAPTER 4

Contents

4.1 data_specification

4.1.1 data_specification package

4.1.1.1 Subpackages

data_specification.enums package

Module contents

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

Bases: enum.Enum

Arithmetic Operations

ADD = 0

Addition operation

MULTIPLY = 2

Multiplication operation

SUBTRACT = 1

Subtraction operation

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
Halts spec execution with an error.

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
Place a value in a register, from an immediate or another register.

NOP = 1
No operation. Can be used as a filler.

PRINT_STRUCT = 130

PRINT_TXT = 129

PRINT_VAL = 128

READ = 65

READ_PARAM = 115

REFORMAT = 106

RESERVE = 2
Reserves a block of memory ready for filling.

SET_WR_PTR = 100
Move the write pointer to a new location, either relative to the start of this reserved memory area or relative to the current write pointer.

START_CONSTRUCTOR = 32

START_STRUCT = 16

STRUCT_ELEM = 17

SWITCH_FOCUS = 80
Swap between different reserved memory regions to work on several at the same time.

WRITE = 66
Performs a simple write or block write operation.

WRITE_ARRAY = 67
Performs a write from an array.

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
Compare the operands for equality

GREATER_THAN = 5
True if the first operand is > the second

GREATER_THAN_OR_EQUAL = 4
True if the first operand is >= the second

LESS_THAN = 3
True if the first operand is < the second

LESS_THAN_OR_EQUAL = 2
True if the first operand is <= the second

NOT_EQUAL = 1
Compare the operands for inequality

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

Supported data types. Internally, these are actually tuples.

1. an identifier for the enum class;
2. the size in bytes of the type;
3. the minimum possible value for the type;
4. the maximum possible value for the type;
5. the scale of the input value to convert it in integer;
6. the pattern to use following the struct package encodings to convert the data in binary format;
7. is whether to apply the scaling when converting to SpiNNaker's binary format.
8. the corresponding numpy type (or None to inhibit direct conversion via numpy, scaled conversion still supported);
9. the text description of the type.

Note: Some types (notably 64-bit fixed-point and floating-point types) are not recommended for use on SpiNNaker due to complications with representability and lack of hardware/library support.

FLOAT_32 = 14
32-bit floating point number

FLOAT_64 = 15
64-bit floating point number (use *not* recommended: hardware/library support inadequate)

INT16 = 5
16-bit signed integer

INT32 = 6
32-bit signed integer

INT64 = 7
64-bit signed integer

INT8 = 4
8-bit signed integer

S015 = 21
0.15 signed fixed point number

S031 = 22
0.32 signed fixed point number

S063 = 23
0.63 signed fixed point number (use *not* recommended: representability)

S07 = 20
0.7 signed fixed point number

S1615 = 12
16.15 signed fixed point number

S3231 = 13
32.31 signed fixed point number (use *not* recommended: representability)

S87 = 11
8.7 signed fixed point number

U016 = 17
0.16 unsigned fixed point number

U032 = 18
0.32 unsigned fixed point number

U064 = 19
0.64 unsigned fixed point number (use *not* recommended: representability)

U08 = 16
0.8 unsigned fixed point number

U1616 = 9
16.16 unsigned fixed point number

U3232 = 10
32.32 unsigned fixed point number (use *not* recommended: representability)

U88 = 8
8.8 unsigned fixed point number

UINT16 = 1
16-bit unsigned integer

UINT32 = 2
32-bit unsigned integer

UINT64 = 3
64-bit unsigned integer

UINT8 = 0
8-bit unsigned integer

decode_array(*values*)
Decodes a byte array into iterable of this type.

Parameters **values** – the bytes to decode into this given data type

Return type numpy array

decode_numpy_array(*array*)
Decode the numpy array of SpiNNaker values according to this type.

Parameters **array** (*ndarray (uint32)*) –

Return type ndarray(uint32 or float64)

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

Parameters **value** (*float or int*) –

Return type bytes

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

Parameters **value** (*float or int*) –

Return type int

encode_as_numpy_int(*value*)
Returns the value as a numpy integer, according to this type.

Parameters **value** (*float or int*) –

Return type uint32

encode_as_numpy_int_array(*array*)
Returns the numpy array as an integer numpy array, according to this type.

Parameters **array** (*ndarray*) –

Return type ndarray

max
The maximum possible value for the type.

Return type Decimal

min
The minimum possible value for the type.

Return type Decimal

numpy_typename
The corresponding numpy type, if one exists.

scale
The scale of the input value to convert it in integer.

Return type Decimal

size
The size in bytes of the type.

Return type int

struct_encoding
The encoding string used for struct. Scaling may also be required.

Return type str

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

Logic Operations

AND = 3
Logical AND

LEFT_SHIFT = 0
Shift left (with zero extension)

NOT = 5
Logical NOT (single argument)

OR = 2
Logical OR

RIGHT_SHIFT = 1
Shift right (with zero extension)

XOR = 4
Logical XOR

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

Random number generator types

MERSENNE_TWISTER = 0
The well-known Mersenne Twister PRNG

data_specification.spi package

Module contents

The interface used by implementations of the executor.

class data_specification.spi.**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*)
This command moves the write pointer to be at the start of the next word if it isn't already at the start of a word.

Implements *ALIGN_WR_PTR*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_arith_op(*cmd*)

This command performs an arithmetic operation.

Implements *ARITH_OP*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_block_copy(*cmd*)

This command copies a block of memory from one location to another.

Implements *BLOCK_COPY*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_break(*cmd*)

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

Implements *BREAK*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_break_loop(*cmd*)

This command stops a loop early.

Implements *BREAK_LOOP*

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

Returns No value returned

Raises

- **DataSpecificationSyntaxError** – If there is an error in the command syntax
- **UnimplementedDSECommandError** – If the command is not implemented.

execute_construct(*cmd*)

This command calls a function.

Implements *CONSTRUCT*

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

Returns No value returned

Raises

- **DataSpecificationSyntaxError** – If there is an error in the command syntax
- **UnimplementedDSECommandError** – If the command is not implemented.

execute_copy_param(*cmd*)

This command copies a field of a structure to another field of a possibly-different structure.

Implements *COPY_PARAM*

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

Returns No value returned

Raises

- **DataSpecificationSyntaxError** – If there is an error in the command syntax
- **UnimplementedDSECommandError** – If the command is not implemented.

execute_copy_struct(*cmd*)

This command copies a structure from one slot to another.

Implements *COPY_STRUCT*

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

Returns No value returned

Raises

- **DataSpecificationSyntaxError** – If there is an error in the command syntax
- **UnimplementedDSECommandError** – If the command is not implemented.

execute_declare_rng(*cmd*)

This command declares a random number generator.

Implements *DECLARE_RNG*

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

Returns No value returned

Raises

- **DataSpecificationSyntaxError** – If there is an error in the command syntax
- **UnimplementedDSECommandError** – If the command is not implemented.

execute_else(*cmd*)

This command handles the other branch of a conditional.

Implements *ELSE*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_end_constructor(*cmd*)

This command ends the definition of a function.

Implements *END_CONSTRUCTOR*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_end_if(*cmd*)

This command ends a conditional.

Implements *END_IF*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_end_loop(*cmd*)

This command finishes a loop.

Implements *END_LOOP*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_end_spec(*cmd*)

This command marks the end of the specification program.

Implements *END_SPEC*

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

Returns A special marker to signal the end.

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_end_struct(*cmd*)

This command completes the definition of a structure.

Implements *END_STRUCT*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_free(*cmd*)

This command frees some memory.

Implements *FREE*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_get_random_rumber(*cmd*)

This command obtains a random number from a distribution.

Implements *GET_RANDOM_NUMBER*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_get_wr_ptr(*cmd*)

This command gets the current write pointer.

Implements *GET_WR_PTR*

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

Returns No value returned

Raises

- **DataSpecificationSyntaxError** – If there is an error in the command syntax
- **UnimplementedDSECommandError** – If the command is not implemented.

execute_if(*cmd*)

This command does a conditional branch.

Implements *IF*

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

Returns No value returned

Raises

- **DataSpecificationSyntaxError** – If there is an error in the command syntax
- **UnimplementedDSECommandError** – If the command is not implemented.

execute_logic_op(*cmd*)

This command performs a logical operation.

Implements *LOGIC_OP*

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

Returns No value returned

Raises

- **DataSpecificationSyntaxError** – If there is an error in the command syntax
- **UnimplementedDSECommandError** – If the command is not implemented.

execute_loop(*cmd*)

This command starts a loop.

Implements *LOOP*

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

Returns No value returned

Raises

- **DataSpecificationSyntaxError** – If there is an error in the command syntax
- **UnimplementedDSECommandError** – If the command is not implemented.

execute_mv(*cmd*)

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

Implements *MV*

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

Returns No value returned

Raises

- **DataSpecificationSyntaxError** – If there is an error in the command syntax
- **UnimplementedDSECommandError** – If the command is not implemented.

execute_nop(*cmd*)

This command executes no operation.

Implements *NOP*

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

Returns No value returned

Return type None

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

execute_print_struct(*cmd*)

This command prints a structure to the log.

Implements *PRINT_STRUCT*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_print_txt(*cmd*)

This command prints a short string to the log.

Implements *PRINT_TXT*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_print_val(*cmd*)

This command prints a value to the log.

Implements *PRINT_VAL*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_random_dist(*cmd*)

This command defines a random disribution.

Implements *DECLARE_RANDOM_DIST*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_read(*cmd*)

This command reads a word from memory.

Implements *READ*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_read_param(*cmd*)

This command extracts an element from a structure.

Implements *READ_PARAM*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_reformat(*cmd*)

This command is never generated!

Implements *REFORMAT*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_reserve(*cmd*)

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

Implements *RESERVE*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax

- **UnimplementedDSECommandError** – If the command is not implemented.

execute_reset_wr_ptr(*cmd*)

This command resets the current write pointer to the beginning of the memory region.

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

Returns No value returned

Raises

- **DataSpecificationSyntaxError** – If there is an error in the command syntax
- **UnimplementedDSECommandError** – If the command is not implemented.

execute_set_wr_ptr(*cmd*)

This command sets the current write pointer.

Implements *SET_WR_PTR*

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

Returns No value returned

Raises

- **DataSpecificationSyntaxError** – If there is an error in the command syntax
- **UnimplementedDSECommandError** – If the command is not implemented.

execute_start_constructor(*cmd*)

This command starts the definition of a function.

Implements *START_CONSTRUCTOR*

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

Returns No value returned

Raises

- **DataSpecificationSyntaxError** – If there is an error in the command syntax
- **UnimplementedDSECommandError** – If the command is not implemented.

execute_start_struct(*cmd*)

This command starts to define a structure.

Implements *START_STRUCT*

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

Returns No value returned

Raises

- **DataSpecificationSyntaxError** – If there is an error in the command syntax
- **UnimplementedDSECommandError** – If the command is not implemented.

execute_struct_elem(*cmd*)

This command adds an element to a structure.

Implements *STRUCT_ELEM*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_switch_focus(*cmd*)

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

Implements *SWITCH_FOCUS*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

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.

Implements *WRITE*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_write_array(*cmd*)

This command writes an array of values in the specified region.

Implements *WRITE_ARRAY*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_write_param(*cmd*)

This command handles a single element of a structure.

Implements *WRITE_PARAM*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_write_param_component(*cmd*)

This command is never generated!

Implements *WRITE_PARAM_COMPONENT*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

execute_write_struct(*cmd*)

This command writes a structure to memory.

Implements *WRITE_STRUCT*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.

4.1.1.2 Submodules

4.1.1.3 data_specification.constants module

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

data_specification.constants.**APPDATA_MAGIC_NUM = 2903706326**

Application data magic number.

data_specification.constants.**APP_PTR_TABLE_BYTE_SIZE = 72**

Size of data spec pointer table produced by DSE, in bytes.

data_specification.constants.**APP_PTR_TABLE_HEADER_BYTE_SIZE = 8**

Size of header of data spec pointer table produced by DSE, in bytes.

data_specification.constants.**DSE_VERSION = 65536**

Version of the file produced by the DSE.

data_specification.constants.**DSG_MAGIC_NUM = 1534894462**

Data spec magic number.

data_specification.constants.**MAX_CONSTRUCTORS = 16**

Maximum number of functions in DSG virtual machine.

data_specification.constants.**MAX_MEM_REGIONS = 16**

Maximum number of memory regions in DSG virtual machine.

data_specification.constants.**MAX_PACKSPEC_SLOTS = 16**
Maximum number of packing specification slots in DSG virtual machine.

data_specification.constants.**MAX_PARAM_LISTS = 16**
Maximum number of parameter lists in DSG virtual machine.

data_specification.constants.**MAX_RANDOM_DISTS = 16**
Maximum number of random distributions in DSG virtual machine.

data_specification.constants.**MAX_REGISTERS = 16**
Maximum number of registers in DSG virtual machine.

data_specification.constants.**MAX_RNGS = 16**
Maximum number of random number generators in DSG virtual machine.

data_specification.constants.**MAX_STRUCT_ELEMENTS = 255**
Maximum number of structure elements in DSG virtual machine.

data_specification.constants.**MAX_STRUCT_SLOTS = 16**
Maximum number of structure slots in DSG virtual machine.

4.1.1.4 data_specification.exceptions module

exception data_specification.exceptions.**DataSpecificationException**
Bases: 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: 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** (*str*) – file being parsed

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 or int*) – The value specified
- **range_min** (*float or int*) – The minimum allowed value
- **range_max** (*float or int*) – 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** (*str or None*) – 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** (*int*) – The ID of the region that was not allocated.
- **command** (*str*) – The name of the command that was being handled.

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** (*str*) – 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** (*int*) – The number of parameters required by the function

4.1.1.5 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.

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** (*str*) – the directory for the reports folder
- **write_text_specs** (*bool*) – True if a textual version of the specification should be written
- **application_run_time_report_folder** (*str*) – The folder to contain the resulting specification files; if 'TEMP' then a temporary directory is used.

Returns the filename of the data writer and the data specification object

Return type 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

Parameters

- **app_data_base_address** (*int*) – base address for the core
- **region** (*int*) – 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** (*str*) – 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

Return type FileIO or None

4.1.1.6 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 `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 `DataSpecificationExecutor` class.

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.

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.

Main API

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

Raises **IOError** – If a read or write fails

dsef

The executor functions themselves.

execute()

Executes the specification. This will result in a configuration of memory regions being done.

Raises

- **IOError** – If a read or write fails
- ***DataSpecificationException*** – If there is an error when executing the specification
- ***TablePointerOutOfMemoryException*** – 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.

Return type numpy.ndarray

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

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

Return type numpy.ndarray

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* or None

mem_regions
An enumeration of the mapping from region ID to region holder.

Return type iterable(tuple(int, *MemoryRegion* or None))

class data_specification.**DataSpecificationExecutorFunctions**(*spec_reader*, *memory_space*)

Bases: data_specification.spi.abstract_executor_functions.AbstractExecutorFunctions

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

Note: DSG operations not mentioned in this class will cause an error during DSE if used.

Parameters

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

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

Implements *BREAK*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.
- ***ExecuteBreakInstruction*** – Raises the exception to break the execution of the DSE

execute_end_spec(*cmd*)
This command marks the end of the specification program.

Implements *END_SPEC*

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

Returns A special marker to signal the end.

Raises

- **DataSpecificationSyntaxError** – If there is an error in the command syntax
- **UnimplementedDSECommandError** – If the command is not implemented.

execute_mv(*cmd*)

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

Implements *MV*

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

Returns No value returned

Raises

- **DataSpecificationSyntaxError** – If there is an error in the command syntax
- **UnimplementedDSECommandError** – If the command is not implemented.

execute_reserve(*cmd*)

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

Implements *RESERVE*

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

Returns No value returned

Raises

- **DataSpecificationSyntaxError** – If there is an error in the command syntax
- **UnimplementedDSECommandError** – If the command is not implemented.
- **ParameterOutOfBoundsException** – If the requested size of the region is beyond the available memory space

execute_set_wr_ptr(*cmd*)

This command sets the current write pointer.

Implements *SET_WR_PTR*

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

Returns No value returned

Raises

- **DataSpecificationSyntaxError** – If there is an error in the command syntax
- **UnimplementedDSECommandError** – If the command is not implemented.
- **NoRegionSelectedException** – If there is no memory region selected for the set-ptr operation

execute_switch_focus(*cmd*)

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

Implements *SWITCH_FOCUS*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.
- ***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.

Implements *WRITE*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.
- ***NoRegionSelectedException*** – If there is no memory region selected for the write operation
- ***RegionNotAllocatedException*** – If the selected region has not been allocated memory space
- ***NoMoreException*** – If the selected region has not enough available memory to store the required data
- ***UnknownTypeLengthException*** – If the data type size is not 1, 2, 4, or 8 bytes

execute_write_array(*cmd*)

This command writes an array of values in the specified region.

Implements *WRITE_ARRAY*

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

Returns No value returned

Raises

- ***DataSpecificationSyntaxError*** – If there is an error in the command syntax
- ***UnimplementedDSECommandError*** – If the command is not implemented.
- ***NoRegionSelectedException*** – If there is no memory region selected for the write operation
- ***RegionNotAllocatedException*** – If the selected region has not been allocated memory space
- ***NoMoreException*** – If the selected region has not enough available memory to store the required data

mem_regions

The collection of memory regions that can be written to.

Return type *MemoryRegionCollection*

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** (*RawIOBase*) – The object to write the specification to
- **report_writer** (*TextIOBase or None*) – 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 **IOError** – If a write to external storage fails

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 $2^{\text{log\_block\_size}}$), 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 or None*) – The ID of a register where the write pointer will be written to once it has been updated, between 0 and 15, or None if no such writing is to be done

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***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
- ***RegionOutOfBoundsException*** – If the move of the pointer would put it outside of the current region
- ***NoRegionSelectedException*** – If no region has been selected

break_loop()

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

Raises

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

- **IOError** – If a write to external storage fails
- ***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

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If `register_id` is not a valid register ID
  - 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
- ***InvalidOperationException*** – 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(int)*) – A list of structure IDs that will be passed into the function, each between 0 and 15

Raises

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

- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If the function ID is not valid
  - If any of the structure IDs are not valid
- ***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`
- ***WrongParameterNumberException*** – If a function is called with a wrong number of parameters
- ***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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***NotAllocatedException*** – If the random distribution ID was not previously declared
- ***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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails

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

Raises

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

Raises

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

- **IOError** – If a write to external storage fails
- ***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
- ***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` (binary data 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 tuple(bytearray, str)

Raises

- ***ParameterOutOfBoundsException*** –
  - If `data_type` is an integer type, and `data` has a fractional part
  - If `data` would overflow the data type
- ***UnknownTypeException*** – If the data type is not known
- ***InvalidSizeException*** – If the data size is invalid

current_region

The ID of the current DSG region we're writing to. If not yet writing to any region, `None`.

Return type int or None

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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***UnknownTypeException*** – If the `rng_type` is not one of the allowed values
- ***ParameterOutOfBoundsException*** –
  - If the `seed` is too big or too small
  - If the `rng_id` is not in the allowed range
- ***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** (*int*) – ID of the distribution being set up
- **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

Raises

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

define_break()

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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – 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(tuple(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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***NoMoreException*** – If there are no more spaces for new structures
- ***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`
- ***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.

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***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

Raises

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

end_function()

Insert command to mark the end of a function definition

Raises ***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.

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***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)

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – 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

Raises

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

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 `parameter_index` is a register ID containing the ID of the parameter/element to copy

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***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

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

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.

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

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If `register_id` is not a valid register ID
  - 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.

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

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If `register_id` is not a valid register ID
  - 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 not 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` is a register ID

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If `register_id` is not a valid register ID
  - If `operand_is_register` is True and `operand` 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.

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

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If `register_id` is not a valid register ID
  - 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.

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

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If `register_id` is not a valid register ID
  - 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.

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

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If `register_id` is not a valid register ID
  - 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

no_operation()

Insert command to execute nothing

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails

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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***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
- ***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** (*str*) – The character encoding to use for the string. Defaults to ASCII.

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – 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 or int*) –
  - 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 value of type given by `data_type`
- **value_is_register** (*bool*) – Indicates if `value` is a register
- **data_type** (`DataType`) – The type of the data to be printed

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***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
- ***UnknownTypeException*** – If `data_type` is not a valid data type

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.

Raises ***ParameterOutOfBoundsException*** – If `dest_id` is out of range for a register ID

region_sizes

A list of sizes of each region that has been reserved.

Note: The list will include `0` for each non-reserved region.

Return type list(int)

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 or None*) – 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

Raises

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

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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails

- **ParameterOutOfBoundsException** – If the `register_id` is not a valid register ID
- **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** (*int or 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 is a value of the type given by `data_type`
- **data_is_register** (*bool*) – Indicates if `data` is a register ID
- **data_type** (`DataType`) – The type of the data to be assigned

Raises

- **DataUndefinedWriterException** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- **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
- **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 or int
  - 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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***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 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
- ***NotAllocatedException*** – If the structure requested has not been declared
- ***UnknownTypeException*** – If the data type is unknown

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 address is a register ID
- **relative_to_current** (*bool*) – Indicates if address (or the value read from that register when address_is_register is True) is to be added to the current address, or used as an absolute address from the start of the current region

Raises

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

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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If register_id is not a valid register ID
  - if value_is_register is True and value is not a valid register ID
- ***UnknownConditionException*** – If 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(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.

Raises

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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If `counter_register_id` is not a valid register ID
  - If `start_is_register` is True and `start` is not a valid register ID
  - If `end_is_register` is True and `end` 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

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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails

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

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

Raises

- **DataUndefinedWriterException** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- **NoRegionSelectedException** – If no region has been 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

Raises

- **IOError** – If a write to external storage fails
- **NoRegionSelectedException** – If no region has been selected

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 or int*) – 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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If `repeats_is_register` is False, and `repeats` is out of range
  - If `repeats_is_register` is True, and `repeats` 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
- ***UnknownTypeException*** – If the data type is not known
- ***InvalidSizeException*** – If the data size is invalid
- ***NoRegionSelectedException*** – If no region has been selected

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*) – Whether `repeats` identifies a register

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If `structure_id` is not a valid structure ID
  - If `repeats_is_register` is False and `repeats` is not in range
  - If `repeats_is_register` is True and `repeats` is not a valid register ID
- ***NoRegionSelectedException*** – If no region has been selected
- ***RegionExhaustedException*** – If the selected region has no more space

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

Insert command to write a value (once) 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** (*int or float*) – the data to write as a float.
- **data_type** (`DataType`) – the type to convert `data` to

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –
  - If `data_type` is an integer type, and `data` has a fractional part
  - If `data` would overflow the data type
- ***UnknownTypeException*** – If the data type is not known
- ***InvalidSizeException*** – If the data size is invalid
- ***NoRegionSelectedException*** – If no region has been selected

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

Raises

- ***DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- **IOError** – If a write to external storage fails
- ***ParameterOutOfBoundsException*** –

- If `repeats_is_register` is False, and `repeats` is out of range
- If `repeats_is_register` is True, and `repeats` is not a valid register ID
- If `data_register` is not a valid register ID

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

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

Bases: `object`

Memory region storage object.

Parameters

- **`unfilled`** (*`bool`*) – if the region should not be written to by the data specification (i.e., because the vertex uses it as a working data region or an output region)
- **`size`** (*`int`*) – the size of the region, in bytes

`allocated_size`

The size of the region.

Return type int

`increment_write_pointer`(*n_bytes*)

Advance the write pointer.

Parameters **`n_bytes`** (*`int`*) – The number of bytes to advance the pointer by.

`max_write_pointer`

The max point where if written over, it will cause an error.

Return type int

`region_data`

The buffer which holds the data written in this region.

Return type bytearray

`remaining_space`

The space between the current write pointer and the end of the region, which is the number of bytes remaining in the region that can be written.

Return type int

`unfilled`

Whether the region is marked as not fillable; unfilled regions will not contain any data at write time.

Return type bool

`write_pointer`

The position in the buffer where data will be written to next.

Return type int

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

Bases: `object`

Collection of memory regions.

Parameters **`n_regions`** (*`int`*) – The number of regions in the collection.

count_used_regions()

The number of regions in the collection that are used.

Return type int

is_empty(*region*)

Parameters **region** (*int*) – The ID of the region

Return type bool

is_unfilled(*region*)

Parameters **region** (*int*) – The ID of the 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** (*int*) – the region ID to which the test is being ran on

Returns whether the region needs to be written

Return type bool

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

regions

The regions in the collection.

Return type iterable(*MemoryRegion*)

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

E

F

G

I

L

M

N

O

P

R

S

T

U

W

X