Transformer API Manual

Release 0.0.1.dev37

Jonathan M Skelton

Dec 30, 2021

Table of Contents

Chapter 1

Transformer

1.1 Transformer package

1.1.1 Subpackages

Transformer.Analysis package

Submodules

Transformer.Analysis.Structure module

Module contents

Transformer.Framework package

Submodules

Transformer.Framework.BatchIO module

Transformer.Framework.Convenience module

Transformer.Framework.Core module

Transformer.Framework.FilterBases module

Transformer.Framework.Filters module

Transformer.Framework.Utilities module

Module contents

Transformer.IO package

Submodules

Transformer.IO.StructureIO module

Transformer.IO.StructureSetIO module

Module contents

Transformer.Screening package

Submodules

Transformer.Screening.MetadiseCalculator module

Transformer.Screening.TotalEnergyCalculatorBase module

Transformer.Screening.Utilities module

Transformer.Screening.Utilities.**ExportRankedEnergiesToCSV** (*rankedEnergies*, *filePath*, *energyUnits='eV'*)

Transformer.Screening.Utilities.**ImportRankedEnergiesFromCSV** (*filePath*)

Transformer.Screening.Utilities.**PrintRankedEnergies** (*rankedEnergies*, *energyUnits='eV'*, *maxPrint=None*)

Transformer.Screening.Utilities.**RankEnergies** (*totalEnergyGroups*)

Module contents

Transformer.Utilities package

Submodules

Transformer.Utilities.DevelopmentTools module

Transformer.Utilities.IOHelper module

Transformer.Utilities.IOHelper.**ClearDirectory** (*directoryPath*, *removeSubdirectories=False*)

Transformer.Utilities.IOHelper.**OpenForCSVWriter** (*filePath*)

Transformer.Utilities.MultiprocessingHelper module

class Transformer.Utilities.MultiprocessingHelper.**AccumulatorBase**

Bases: object

Base for Accumulators to be passed to the QueueAccumulate() routine.

Accumulate (*item*)

Process/accumulate a new item. This method must be overridden in derived classes.

Parameters **item** (*[type]*) – [description]

Raises **NotImplementedError** – *Accumulate()* must be overridden by derived classes.

Finalise ( )

Finalise processing and return accumulated output. This method must be overridden in derived classes.

Raises **NotImplementedError** – *Finalise()* must be overridden by derived classes.

Transformer.Utilities.MultiprocessingHelper.**CPUCount** ( )

Return the number of CPU cores on the system.

> **Warning** If multiprocessing.cpu_count() raises a *NotImplementedError* (unlikely), this wrapper issues a warning and returns a "safe" value of 1.

Returns [description]

Return type [type]

class Transformer.Utilities.MultiprocessingHelper.**Counter** (*initialValue=0*, *readLock=True*)

Bases: object

Implements a simple shared-memory integer counter. Increments and decrements are protected by a lock, and reads can optionally also be protected.

Current ( )

Return the current value of the counter.

Returns [description]

Return type [type]

Decrement (*amount=1*)

Decrement the counter.

Parameters **amount** (*int, optional*) – amount to subtract from the counter, by default 1.

Increment (*amount=1*)

Increment the counter.

Parameters **amount** (*int, optional*) – amount to add to the counter, by default 1.

class Transformer.Utilities.MultiprocessingHelper.**FunctionMapper** (*mapFunction*)

Bases: Transformer.Utilities.MultiprocessingHelper.MapperBase

Basic Mapper which wrapps a supplied mapping function.

Map (*item*)

Map item to output using the function supplied to the constructor.

Note

- If item is a single value, it is passed to the mapping function using map_function(item); if item is a tuple, it is passed with map_function(*item).
- For functions requiring a single tuple, wrap it in an outer tuple with e.g. ((arg1, arg2),).

Parameters **item** (*[type]*) – [description]

Returns [description]

Return type [type]

class Transformer.Utilities.MultiprocessingHelper.**MapperBase**

Bases: object

Base for Mapper classes to be passed to the QueueMap() routine.

Map (*item*)

Map item and return output. This method must be overridden by derived classes.

Parameters **item** (*[type]*) – [description]

Raises **NotImplementedError** – *Map()* must be overridden in a derived class.

Transformer.Utilities.MultiprocessingHelper.**PollDelay = 0.001**

Per-process number of items used to define the batch size for queue-based inter-process communication.

Transformer.Utilities.MultiprocessingHelper.**QueueAccumulate** (*inputList*, *accumulators*, *progressBar=False*)

Accumulate items in inputList, dividing the work among the supplied set of Accumulator objects. Each Accumulator is passed to a worker process, and the input list is processed in parallel using a queue-based system.

Note

- If only one Accumulator is supplied, the input list will be processed in serial.

Warning

- As for the QueueMap() function, setting *progressBar* = *True* when the tqdm module is not available will issue a warning, and a progress bar will not be displayed.

Parameters

- **inputList** (*list*) – list of inputs to process with Accumulators.
- **accumulators** (*[type]*) – a set of user-defined Accumulator objects; the number supplied sets the number of worker processes spawned.
- **progressBar** (*bool, optional*) – if True, display a progress bar during

mapping (requires the *tqdm* module).

Returns **accumulatorResults** – [description]

Return type list

Raises
- **`AssertionError`** – if inputList is None.
- **`Exception`** – if no accumulator is supplied.

`Transformer.Utilities.MultiprocessingHelper.`**`QueueMap`**(*inputList*, *mappers*, *progressBar=False*)

Map items in inputList to an in-order list of outputs, dividing the work among the supplied set of Mapper objects. Each Mapper is passed to a worker process, and the input list is processed in parallel using a queue-based producer-consumer model.

Note

- There is no guarentee which *Mapper* will process which input item(s), so all Mappers must return the same result for a given input.
- The reason for using *Mapper* objects rather than a single mapping function is so each *Mapper* can e.g. use different working directories.
- If the flexibilty of Mappers is not needed, the `QueueMapFunction()` routine presents a similar interface to the `map()` function.

Warning

- If only one mapper is supplied, the input list will be mapped in serial.
- If the `tqdm` module is not available, setting *progressBar = True* will issue a warning and a progress bar will not be displayed.

Parameters
- **`inputList`** (*`list`*) – list of inputs to process with the Mappers.
- **`mappers`** (*`[type]`*) – a set of user-defined Mapper objects; the number of Mappers sets the number of worker processes that will be spawned.
- **`progressBar`** (*`bool, optional`*) – if True, and if the tqdm module is available, display a progress bar during mapping, by default False.

Returns [description]

Return type [type]

Raises
- **`Exception`** – [description]
- **`Exception`** – [description]

`Transformer.Utilities.MultiprocessingHelper.`**`QueueMapFunction`**(*mapFunction*, *inputList*, *maxNumProcesses=8*, *progressBar=True*)

Map items in inputList through mapFunction and return a list of outputs. This routine effectively implements a queue-based alternative to `map()` with support for a TQDM progress bar.

Note

- Internally, *mapFunction* is wrapped by `FunctionMapper` classes; therefore, pasing input items to the function works as per the `Map()` function of *FunctionMapper*.

- If an item is a single value, it is passed to the mapping function with *map_function(item)*; if it is a tuple, it is passed as *map_function(*item)*.
- Single-tuple arguments will need to be wrapped in an outer tuple, e.g. *((arg1, arg2),)*.
- As for `QueueMap()`, if *maxNumProcesses* is set to 1, a serial mapping will be performed without spawning any worker processes.

Warning

- Similarly, if the `tqdm` module is not available, setting *progressBar = True* will not work and will cause a warning to be issued.

Parameters
- **mapFunction** (*[type]*) – [description]
- **inputList** (*[type]*) – [description]
- **maxNumProcesses** (*[type], optional*) – maximum number of worker processes, by default MultiprocessingHelper.CPUCount().
- **progressBar** (*bool, optional*) – if True, and if the tqdm module is available, display a progress bar during mapping, by default True.

Returns [description]

Return type [type]

Raises **AssertionError** – if mapFunction is None.

`Transformer.Utilities.MultiprocessingHelper.`**`_QueueAccumulate_ProcessMain`**(*accumulator*, *inputQueue*, *inputCounter*, *outputQueue*, *terminateFlag*)

Worker process function for processes spawned by the `QueueAccumulate()` function.

Parameters
- **accumulator** (*[type]*) – Accumulator object to be used to accumulate input items.
- **inputQueue** (*[type]*) – queue from which to retrieve input items to process.
- **inputCounter** (*[type]*) – shared-memory counter used to track the progress of the input processing.
- **outputQueue** (*[type]*) – queue in which to place the result returned by the *Finalise()* method of the accumulator once all input items have been processed.
- **terminateFlag** (*[type]*) – shared-memory flag used to signal the worker process to finalise the accumulation, return the result, and terminate.

`Transformer.Utilities.MultiprocessingHelper.`**`_QueueMap_ProcessMain`**(*mapper*, *inputQueue*, *outputQueue*, *terminateFlag*)

Worker process function for processes spawned by the `MultiprocessingHelperQueueMap()` function.

Parameters
- **mapper** (*[type]*) – Mapper object to be used to map input items to outputs.
- **inputQueue** (*[type]*) – queue from which to retrieve (index, item) tuples to process.
- **outputQueue** (*[type]*) – queue in which to place (index, item) output.
- **terminateFlag** (*[type]*) – shared-memory flag used to signal the worker process to terminate.

Transformer.Utilities.StructureTools module

Transformer.Utilities.StructureTools.**CartesianToFractionalCoordinates** (*latticeVectors, atomPositions*)

Transformer.Utilities.StructureTools.**PrintStructureSetSummary** (*structureSet*)

Module contents

1.1.2 Submodules

Transformer.Constants module

Contains constants used by other modules.

Transformer.Constants.**AtomicNumberToSymbol** (*atomicNumber*)
Lookup atomicNumber in the periodic table and return the corresponding atomic symbol.

Transformer.Constants.**SymbolToAtomicNumber** (*symbol*)
Lookup symbol in the periodic table and return the corresponding atomic number.

Transformer.Structure module

Transformer.StructureSet module

1.1.3 Module contents

- genindex
- modindex
- search

m

t

Symbols

A

C

D

E

F

I

M

O

P

Q

R

S

T