

Base

Geometry

Class representing geometry of an atom arrangement.

Attributes:

Name	Type	Description
<code>sites</code>	<code>List[Tuple[float, float]]</code>	Atom site arrangement
<code>filling</code>	<code>List[int]</code>	Which sites are filled
<code>parallel_decoder</code>	<code>Optional[ParallelDecoder]</code>	Decoder object for decoding Geometry object

LocalTask

Bases: `Task`

`Task` to use for local executions for simulation purposes..

RemoteTask

Bases: `Task`

`Task` to use for remote executions to run the program on Quera Quantum Computers.

Report

```
Report(data, metas, geos, name='')
```

Report is a helper class for organizing and analysing data

ANALYZING RESULTS

When you've retrieved your results from either emulation or hardware you can generate a

`.report()`:

```
report = results.report()
```

For the examples below we analyze the results of a two atom program.

The report contains useful information such as:

- The raw bitstrings measured per each execution of the program

```
>>> report.bitstrings()
[array([[1, 1],
        [1, 1],
        [1, 1],
        ...,
        [1, 1],
        [1, 1],
```

- The number of times each unique bitstring occurred:

```
>>> report.counts()
[OrderedDict([('11', 892), ('10', 59), ('01', 49)])]
```

- The Rydberg Density for each atom

```
>>> report.rydberg_densities()

           0      1
task_number
0          0.053  0.054
```

” Source code in `src\bloqade\task\base.py`

```
160 def __init__(self, data, metas, geos, name="") -> None:
161     self.dataframe = data # df
162     self._bitstrings = None # bitstring cache
163     self._counts = None # counts cache
164     self.metas = metas
165     self.geos = geos
166     self.name = name + " " + str(datetime.datetime.now())
```

markdown property

```
markdown
```

Get the markdown representation of the dataframe

bitstrings

```
bitstrings(filter_perfect_filling=True, clusters=[])
```

Get the bitstrings from the data.

Parameters:

Name	Type	Description	Default
<code>filter_perfect_filling</code>	<code>bool</code>	whether return will only contain perfect filling shots. Defaults to True.	<code>True</code>
<code>clusters</code>	<code>Union[tuple[int, int], List[tuple[int, int]]]</code>	<code>(tuple[int, int], Sequence[Tuple[int, int]]):</code> cluster index to filter shots from. If none are provided all clusters are used, defaults to [].	<code>[]</code>

Returns:

Name	Type	Description
<code>bitstrings</code>	<code>list of ndarray</code>	list corresponding to each task in the report. Each element is an ndarray of shape (nshots, nsites) where nshots is the number of shots for the task and nsites is the number of sites in the task. For example: <pre>[array([[1, 1], [1, 1],</pre>

Name	Type	Description
		<pre>[1, 1], ..., [1, 1], [1, 1], [1, 0]], dtype=int8]</pre>

Note

Note that nshots may vary between tasks if `filter_perfect_filling` is set to True.

Source code in `src\bloqade\task\base.py`

```

220 @beartype
221 def bitstrings(
222     self,
223     filter_perfect_filling: bool = True,
224     clusters: Union[tuple[int, int], List[tuple[int, int]]] = [],
225 ) -> List[NDArray]:
226     """Get the bitstrings from the data.
227
228     Args:
229     filter_perfect_filling (bool): whether return will
230         only contain perfect filling shots. Defaults to True.
231     clusters: (tuple[int, int], Sequence[Tuple[int, int]]):
232         cluster index to filter shots from. If none are provided
233         all clusters are used, defaults to [].
234
235     Returns:
236     bitstrings (list of ndarray): list corresponding to each
237         task in the report. Each element is an ndarray of shape
238         (nshots, nsites) where nshots is the number of shots for
239         the task and nsites is the number of sites in the task.
240         For example:
241         ```python3
242         [array([[1, 1],
243                [1, 1],
244                [1, 1],
245                ...,
246                [1, 1],
247                [1, 1],
248                [1, 0]], dtype=int8)]
249         ```
250
251     Note:
252     Note that nshots may vary between tasks if filter_perfect_filling
253     is set to True.
254
255     """
256
257     task_numbers =
258     self.dataframe.index.get_level_values("task_number").unique()
259
260     bitstrings = []
261     for task_number in task_numbers:
262         mask = self._filter(
263             task_number=task_number,
264             filter_perfect_filling=filter_perfect_filling,
265             clusters=clusters,
266         )
267         if np.any(mask):
268             bitstrings.append(self.dataframe.loc[mask].to_numpy())
269         else:
270             bitstrings.append(
271                 np.zeros((0, self.dataframe.shape[1]), dtype=np.uint8)
272             )

```

```

272
273     return bitstrings

```

counts

```
counts(filter_perfect_filling=True, clusters=[])
```

Get the counts of unique bit strings.

Parameters:

Name	Type	Description	Default
<code>filter_perfect_filling</code>	<code>bool</code>	whether return will only contain perfect filling shots. Defaults to True.	<code>True</code>
<code>clusters</code>	<code>Union[tuple[int, int], List[tuple[int, int]]]</code>	<code>(tuple[int, int], Sequence[Tuple[int, int]]):</code> cluster index to filter shots from. If none are provided all clusters are used, defaults to [].	<code>[]</code>

Returns:

Name	Type	Description
<code>counts</code>	<code>list of OrderedDict[str, int]</code>	list corresponding to each task in the report. Each element is an ndarray of shape (nshots, nsites) where nshots is the number of shots for the task and nsites is the number of sites in the task. For example: <div style="background-color: #f0f0f0; padding: 5px; margin-top: 10px;"> <pre>[OrderedDict([('11', 892), ('10', 59), ('01', 49)])]</pre> </div>

 **Note**

Note that nshots may vary between tasks if `filter_perfect_filling` is set to True.

Source code in `src\bloqade\task\base.py`

```

275 def counts(
276     self,
277     filter_perfect_filling: bool = True,
278     clusters: Union[tuple[int, int], List[tuple[int, int]]] = [],
279 ) -> List[OrderedDict[str, int]]:
280     """Get the counts of unique bit strings.
281
282     Args:
283     filter_perfect_filling (bool): whether return will
284     only contain perfect filling shots. Defaults to True.
285     clusters: (tuple[int, int], Sequence[Tuple[int, int]]):
286     cluster index to filter shots from. If none are provided
287     all clusters are used, defaults to [].
288
289     Returns:
290     counts (list of OrderedDict[str, int]): list corresponding to each
291     task in the report. Each element is an ndarray of shape
292     (nshots, nsites) where nshots is the number of shots for
293     the task and nsites is the number of sites in the task.
294     For example:
295     ```python
296     [OrderedDict([('11', 892), ('10', 59), ('01', 49)])]
297     ```
298
299     Note:
300     Note that nshots may vary between tasks if filter_perfect_filling
301     is set to True.
302
303     """
304
305     def _generate_counts(bitstring):
306         output = np.unique(bitstring, axis=0, return_counts=True)
307
308         count_list = [
309             (".".join(map(str, bitstring)), int(count))
310             for bitstring, count in zip(*output)
311         ]
312         count_list.sort(key=lambda x: x[1], reverse=True)
313         count = OrderedDict(count_list)
314
315         return count
316
317     return list(
318         map(_generate_counts, self.bitstrings(filter_perfect_filling,
319 clusters))
320     )

```

list_param


```
list_param(field_name)
```

List the parameters associate with the given variable `field_name` for each tasks.

Parameters:

Name	Type	Description	Default
<code>field_name</code>	<code>str</code>	variable name	<i>required</i>

” Source code in `src\bloqade\task\base.py`

```

168 def list_param(self, field_name: str) -> List[Union[Number, None]]:
169     """
170     List the parameters associate with the given variable field_name
171     for each tasks.
172
173     Args:
174         field_name (str): variable name
175
176     """
177
178     def cast(x):
179         if x is None:
180             return None
181         elif isinstance(x, (list, tuple, np.ndarray)):
182             return list(map(cast, x))
183         else:
184             return float(x)
185
186     return list(map(cast, (meta.get(field_name) for meta in self.metas)))

```

rydberg_densities

```
rydberg_densities(filter_perfect_filling=True, clusters=[])
```

Get rydberg density for each task.

Parameters:

Name	Type	Description	Default
<code>filter_perfect_filling</code>	<code>bool</code>	whether return will only contain perfect filling shots. Defaults to True.	<code>True</code>
<code>clusters</code>	<code>Union[tuple[int, int], List[tuple[int, int]]]</code>	<code>(tuple[int, int], Sequence[Tuple[int, int]]):</code> cluster index to filter shots from. If none are provided all clusters are used, defaults to <code>[]</code> .	<code>[]</code>

Returns:

Name	Type	Description
<code>rydberg_densities</code>	<code>Union[Series, DataFrame]</code>	per-site rydberg density for each task as a pandas DataFrame or Series. For example: <div style="background-color: #f0f0f0; padding: 5px; margin-top: 10px;"> <pre> 0 1 task_number 0 0.053 0.054 </pre> </div>

Source code in `src\bloqade\task\base.py`

```

321 @beartype
322 def rydberg_densities(
323     self,
324     filter_perfect_filling: bool = True,
325     clusters: Union[tuple[int, int], List[tuple[int, int]]] = [],
326 ) -> Union[pd.Series, pd.DataFrame]:
327     """Get rydberg density for each task.
328
329     Args:
330         filter_perfect_filling (bool, optional): whether return will
331             only contain perfect filling shots. Defaults to True.
332         clusters: (tuple[int, int], Sequence[Tuple[int, int]]):
333             cluster index to filter shots from. If none are provided
334             all clusters are used, defaults to [].
335
336     Returns:
337         rydberg_densities (Union[pd.Series, pd.DataFrame]):
338             per-site rydberg density for each task as a pandas DataFrame or
339             Series.
340         For example:
341         ```python
342         0 1
343         task_number
344         0 0.053 0.054
345         ```
346     """
347     mask = self._filter(
348         filter_perfect_filling=filter_perfect_filling, clusters=clusters
349     )
350     df = self.dataframe[mask]
351     return 1 - (df.groupby("task_number").mean())

```

show

show()

Interactive Visualization of the Report

Source code in `src\bloqade\task\base.py`

```
352     def show(self):
353         """
354         Interactive Visualization of the Report
355         """
356         display_report(self)
```