Braket

Helper functions related to IR submission co-ordinations between Bloqade and Braket

BraketTaskSpecification

Bases: `BaseModel`

Class representing geometry of an atom arrangement.

Attributes:

Name	Type	Description
`nshots`	`int`	Number of shots
`program`	`Program`	IR(Intermediate Representation) program suitable for braket

extract_braket_program

```
extract_braket_program(quera_task_ir)
```

Extracts the Braket program.

Parameters:

Name	Type	Description	Default
`quera_task_ir`	`QuEraTaskSpecification`	Quera IR(Intermediate representation) of the task.	*required*

Source code in `src\bloqade\submission\ir\braket.py`

```
def extract_braket_program(quera_task_ir: QuEraTaskSpecification):
    """Extracts the Braket program.

    Args:
        quera_task_ir (QuEraTaskSpecification):
            Quera IR(Intermediate representation) of the task.
    """
    lattice = quera_task_ir.lattice

    rabi_amplitude = (

quera_task_ir.effective_hamiltonian.rydberg.rabi_frequency_amplitude.global_
    )
    rabi_phase = (

quera_task_ir.effective_hamiltonian.rydberg.rabi_frequency_phase.global_
    )
    global_detuning =
quera_task_ir.effective_hamiltonian.rydberg.detuning.global_
    local_detuning =
quera_task_ir.effective_hamiltonian.rydberg.detuning.local

    register = AtomArrangement()
    for site, filled in zip(lattice.sites, lattice.filling):
        site_type = SiteType.FILLED if filled == 1 else SiteType.VACANT
        register.add(site, site_type)

    hamiltonian = DrivingField(
        amplitude=to_braket_field(rabi_amplitude),
        phase=to_braket_field(rabi_phase),
        detuning=to_braket_field(global_detuning),
    )

    if local_detuning:
        hamiltonian = hamiltonian +
ShiftingField(to_braket_field(local_detuning))

    return AnalogHamiltonianSimulation(
        register=register,
        hamiltonian=hamiltonian,
    )
```

from_braket_status_codes

```
from_braket_status_codes(braket_status)
```

Gets the `QuEraTaskStatusCode` object for working with Bloqade SDK.

Parameters:

Name	Type	Description	Default
`braket_status`	`str`	str The value of status in metadata() in the Amazon Braket. `GetQuantumTask` operation. If use_cached_value is True, the value most recently returned from `GetQuantumTask` operation is used	*required*

Returns:

Type	Description
`QuEraTaskStatusCode`	An object of the type `Field` in Braket SDK

Source code in `src\bloqade\submission\ir\braket.py`

```python
def from_braket_status_codes(braket_status: str) -> QuEraTaskStatusCode:
    """Gets the `QuEraTaskStatusCode` object for working with Bloqade SDK.

    Args:
        braket_status: str
            The value of status in metadata() in the Amazon Braket.
            `GetQuantumTask` operation. If use_cached_value is True,
            the value most recently returned from
            `GetQuantumTask` operation is used

    Returns:
        An object of the type `Field` in Braket SDK
    """
    if braket_status == str("QUEUED"):
        return QuEraTaskStatusCode.Enqueued
    else:
        return QuEraTaskStatusCode(braket_status.lower().capitalize())
```

from_braket_task_results

```
from_braket_task_results(braket_task_results)
```

Get the `QuEraTaskResults` object for working with Bloqade SDK.

Parameters:

Name	Type	Description
`braket_task_results`	`AnalogHamiltonianSimulationTaskResult`	AnalogHamiltonianSimulati Quantum task result of bral

Returns:

Type	Description
`QuEraTaskResults`	An object of the type `Field` in Braket SDK.

Source code in `src\bloqade\submission\ir\braket.py`

```python
def from_braket_task_results(
    braket_task_results: AnalogHamiltonianSimulationTaskResult,
) -> QuEraTaskResults:
    """Get the `QuEraTaskResults` object for working with Bloqade SDK.

    Args:
        braket_task_results: AnalogHamiltonianSimulationTaskResult
            Quantum task result of braket system

    Returns:
        An object of the type `Field` in Braket SDK.
    """
    shot_outputs = []
    for measurement in braket_task_results.measurements:
        shot_outputs.append(
            QuEraShotResult(
                shot_status=QuEraShotStatusCode.Completed,
                pre_sequence=list(measurement.pre_sequence),
                post_sequence=list(measurement.post_sequence),
            )
        )

    return QuEraTaskResults(
        task_status=QuEraTaskStatusCode.Completed, shot_outputs=shot_outputs
    )
```

to_braket_field

```
to_braket_field(quera_field)
```

Converts to `TimeSeries` object supported by Braket.

Parameters:

Name	Type	Description	Default
`quera_field`	`Union[GlobalField, LocalField)]`	Field supported by Quera	*required*

Returns:

Type	Description
`Field`	An object of the type `braket.ahs.field.Field`

Raises:

Type	Description
`TypeError`	If field is not of the type `GlobalField` or `LocalField`.

Source code in `src\bloqade\submission\ir\braket.py`

```python
def to_braket_field(quera_field: Union[GlobalField, LocalField]) -> Field:
    """Converts to `TimeSeries` object supported by Braket.

    Args:
        quera_field (Union[GlobalField, LocalField)]:
            Field supported by Quera

    Returns:
        An object of the type `braket.ahs.field.Field`

    Raises:
        TypeError: If field is not of the type `GlobalField` or `LocalField`.
    """
    if isinstance(quera_field, GlobalField):
        times = quera_field.times
        values = quera_field.values
        time_series = to_braket_time_series(times, values)
        return Field(pattern="uniform", time_series=time_series)
    elif isinstance(quera_field, LocalField):
        times = quera_field.times
        values = quera_field.values
        pattern = quera_field.lattice_site_coefficients
        time_series = to_braket_time_series(times, values)
        pattern = Pattern(pattern)
        return Field(pattern=pattern, time_series=time_series)
    else:
        raise TypeError
```

to_braket_task

```
to_braket_task(quera_task_ir)
```

Converts to `Tuple[int, AnalogHamiltonianSimulation]` object supported by Braket.

Parameters:

Name	Type	Description	Default
`quera_task_ir`	`QuEraTaskSpecification`	Quera IR(Intermediate representation) of the task.	*required*

Returns:

Type	Description
`Tuple[int, AnalogHamiltonianSimulation]`	An tuple of the type `Tuple[int, AnalogHamiltonianSimulation]`.

Source code in `src\bloqade\submission\ir\braket.py`

```
def to_braket_task(
    quera_task_ir: QuEraTaskSpecification,
) -> Tuple[int, AnalogHamiltonianSimulation]:
    """Converts to `Tuple[int, AnalogHamiltonianSimulation]` object supported
by Braket.

    Args:
        quera_task_ir (QuEraTaskSpecification):
            Quera IR(Intermediate representation) of the task.

    Returns:
        An tuple  of the type `Tuple[int, AnalogHamiltonianSimulation]`.
    """
    braket_ahs_program = extract_braket_program(quera_task_ir)
    return quera_task_ir.nshots, braket_ahs_program
```

to_braket_task_ir

```
to_braket_task_ir(quera_task_ir)
```

Converts quera IR(Intermendiate Representation) to to `BraketTaskSpecification` object.

Parameters:

Name	Type	Description	Default
`quera_task_ir`	`QuEraTaskSpecification`	Quera IR(Intermediate representation) of the task.	*required*

Returns:

Type	Description
`BraketTaskSpecification`	An object of the type `BraketTaskSpecification` in Braket SDK

Source code in `src\bloqade\submission\ir\braket.py`

```python
def to_braket_task_ir(quera_task_ir: QuEraTaskSpecification) -> 
BraketTaskSpecification:
    """Converts quera IR(Intermendiate Representation) to
    to `BraketTaskSpecification` object.

    Args:
        quera_task_ir (QuEraTaskSpecification):
            Quera IR(Intermediate representation) of the task.

    Returns:
        An object of the type `BraketTaskSpecification` in Braket SDK

    """
    nshots, braket_ahs_program = to_braket_task(quera_task_ir)
    return BraketTaskSpecification(nshots=nshots,
program=braket_ahs_program.to_ir())
```

to_braket_time_series

```
to_braket_time_series(times, values)
```

Converts to `TimeSeries` object supported by Braket.

Parameters:

Name	Type	Description	Default
`times`	`List[Decimal]`	Times of the value.	*required*
`values`	`List[Decimal]`	Corresponding values to add to the time series	*required*

Returns:

Type	Description
`TimeSeries`	An object of the type `braket.timings.TimeSeries`

Source code in `src\bloqade\submission\ir\braket.py`

```python
def to_braket_time_series(times: List[Decimal], values: List[Decimal]) ->
TimeSeries:
    """Converts to `TimeSeries` object supported by Braket.

    Args:
        times (List[Decimal]): Times of the value.
        values (List[Decimal]): Corresponding values to add to the time series

    Returns:
        An object of the type `braket.timings.TimeSeries`
    """
    time_series = TimeSeries()
    for time, value in zip(times, values):
        time_series.put(time, value)

    return time_series
```

to_quera_capabilities

```
to_quera_capabilities(paradigm)
```

Converts to `QuEraCapabilities` object supported by Braket.

Parameters:

Name	Type	Description	Default
`paradigm`		Bracket paradigm	*required*

Returns:

Type	Description
`QuEraCapabilities`	An object of the type `QuEraCapabilities` in Bloqade SDK.

Source code in `src\bloqade\submission\ir\braket.py`

```
def to_quera_capabilities(paradigm) -> cp.QuEraCapabilities:
    """Converts to `QuEraCapabilities` object supported by Braket.

    Args:
        paradigm: Bracket paradigm

    Returns:
        An object of the type `QuEraCapabilities` in Bloqade SDK.
    """
    rydberg_global = paradigm.rydberg.rydbergGlobal

    return cp.QuEraCapabilities(
        version=paradigm.braketSchemaHeader.version,
        capabilities=cp.DeviceCapabilities(
            task=cp.TaskCapabilities(
                number_shots_min=1,
                number_shots_max=1000,
            ),
            lattice=cp.LatticeCapabilities(
                number_qubits_max=paradigm.qubitCount,
                geometry=cp.LatticeGeometryCapabilities(

spacing_radial_min=paradigm.lattice.geometry.spacingRadialMin,

spacing_vertical_min=paradigm.lattice.geometry.spacingVerticalMin,

position_resolution=paradigm.lattice.geometry.positionResolution,

number_sites_max=paradigm.lattice.geometry.numberSitesMax,
                ),
                area=cp.LatticeAreaCapabilities(
                    width=paradigm.lattice.area.width,
                    height=paradigm.lattice.area.height,
                ),
            ),
            rydberg=cp.RydbergCapabilities(
                c6_coefficient=paradigm.rydberg.c6Coefficient,
                global_=cp.RydbergGlobalCapabilities(
                    rabi_frequency_max=rydberg_global.rabiFrequencyRange[0],
                    rabi_frequency_min=rydberg_global.rabiFrequencyRange[1],

rabi_frequency_resolution=rydberg_global.rabiFrequencyResolution,

rabi_frequency_slew_rate_max=rydberg_global.rabiFrequencySlewRateMax,
                    detuning_max=rydberg_global.detuningRange[0],
                    detuning_min=rydberg_global.detuningRange[1],
                    detuning_resolution=rydberg_global.detuningResolution,

detuning_slew_rate_max=rydberg_global.detuningSlewRateMax,
                    phase_min=rydberg_global.phaseRange[0],
                    phase_max=rydberg_global.phaseRange[1],
                    phase_resolution=rydberg_global.phaseResolution,
                    time_min=rydberg_global.timeMin,
                        time_max=rydberg_global.timeMax,
                        time_resolution=rydberg_global.timeResolution,
                        time_delta_min=rydberg_global.timeDeltaMin,
                    ),
                    local=None,
                ),
            ),
        )
```