coreform.job

Table of contents

Source directory: `interop/abaqus/cf_app/coreform/job`

Directory tree

- `job/`

1 artifacts.py

Module path: `coreform.job.artifacts`

Available API members

Name	Kind	Description
`AbaqusArtifactPaths.prefix`	Attribute	Basename shared by the submitted Job and its primary artifacts.
`AbaqusArtifactPaths.mesh_cf`	Attribute	Coreform input file consumed by the Mesh stage.
`AbaqusArtifactPaths.interop_cf`	Attribute	Coreform input file consumed by the Abaqus interop stage.
`AbaqusArtifactPaths.mesh_sq`	Attribute	Mesh database produced by the Mesh stage.
`AbaqusArtifactPaths.interop_sq`	Attribute	Mesh database updated by the interop stage.
`AbaqusArtifactPaths.main_inp`	Attribute	Abaqus input deck translated and submitted under `prefix`.

Name	Kind	Description
AbaqusArtifactPaths.source_inp	Attribute	Protected copy of the untranslated Abaqus input deck.
AbaqusArtifactPaths.geometry_inp	Attribute	Generated input deck containing the IGA geometry.
AbaqusArtifactPaths.keyword_report	Attribute	JSON report describing Abaqus keyword translation.
AbaqusArtifactPaths.completion_msg	Attribute	Abaqus message file used to detect completion.
AbaqusArtifactPaths.results_odb	Attribute	Abaqus output database produced by the solve stage.
AbaqusArtifactPaths.results_sql	Attribute	Coreform results database produced by the solve stage.
abaqus_artifact_paths	Function	Resolve artifact paths shared by the portable plan and Abaqus adapter.
AbaqusArtifactPaths	Class	Resolved relative paths for one portable Abaqus Job pipeline.

1.1 AbaqusArtifactPaths

Object path: coreform.job.artifacts.AbaqusArtifactPaths

Resolved relative paths for one portable Abaqus Job pipeline.

Call syntax

```
result = AbaqusArtifactPaths(prefix=prefix, mesh_cf=mesh_cf,
interop_cf=interop_cf, mesh_sql=mesh_sql, interop_sql=interop_sql,
main_inp=main_inp, source_inp=source_inp, geometry_inp=geometry_inp,
keyword_report=keyword_report, completion_msg=completion_msg,
results_odb=results_odb, results_sql=results_sql)
```

Signatures

```
AbaqusArtifactPaths(prefix: str, mesh_cf: str, interop_cf: str, mesh_sql:
str, interop_sql: str, main_inp: str, source_inp: str, geometry_inp: str,
keyword_report: str, completion_msg: str, results_odb: str, results_sql: str)
```

Parameters

Name	Type	Required	Default	Description
`prefix`	`str`	Yes	—	Basename shared by the submitted Job and its primary artifacts.
`mesh_cf`	`str`	Yes	—	Coreform input file consumed by the Mesh stage.
`interop_cf`	`str`	Yes	—	Coreform input file consumed by the Abaqus interop stage.
`mesh_sql`	`str`	Yes	—	Mesh database produced by the Mesh stage.
`interop_sql`	`str`	Yes	—	Mesh database updated by the interop stage.
`main_inp`	`str`	Yes	—	Abaqus input deck translated and submitted under `prefix`.
`source_inp`	`str`	Yes	—	Protected copy of the untranslated Abaqus input deck.

Name	Type	Required	Default	Description
`geometry_inp`	`str`	Yes	—	Generated input deck containing the IGA geometry.
`keyword_report`	`str`	Yes	—	JSON report describing Abaqus keyword translation.
`completion_msg`	`str`	Yes	—	Abaqus message file used to detect completion.
`results_odb`	`str`	Yes	—	Abaqus output database produced by the solve stage.
`results_sql`	`str`	Yes	—	Coreform results database produced by the solve stage.

1.1.a Attributes

Name	Type	Value	Description
`prefix`	`str`	—	Basename shared by the submitted Job and its primary artifacts.
`mesh_cf`	`str`	—	Coreform input file consumed by the Mesh stage.
`interop_cf`	`str`	—	Coreform input file consumed by the Abaqus interop stage.

Name	Type	Value	Description
`mesh_sql`	`str`	—	Mesh database produced by the Mesh stage.
`interop_sql`	`str`	—	Mesh database updated by the interop stage.
`main_inp`	`str`	—	Abaqus input deck translated and submitted under `prefix`.
`source_inp`	`str`	—	Protected copy of the untranslated Abaqus input deck.
`geometry_inp`	`str`	—	Generated input deck containing the IGA geometry.
`keyword_report`	`str`	—	JSON report describing Abaqus keyword translation.
`completion_msg`	`str`	—	Abaqus message file used to detect completion.
`results_odb`	`str`	—	Abaqus output database produced by the solve stage.
`results_sql`	`str`	—	Coreform results database produced by the solve stage.

1.2 `abaqus_artifact_paths`

Object path: `coreform.job.artifacts.abaqus_artifact_paths`

Description: Resolve artifact paths shared by the portable plan and Abaqus adapter.

Call syntax

```
result = coreform.job.artifacts.abaqus_artifact_paths(value=value)
```

Returns

Type	Description
`AbaqusArtifactPaths`	Deterministically derived filenames for every pipeline artifact.

Signatures

```
abaqus_artifact_paths(value: Union['IGAJob', 'CoreformIGAJobSpec']) ->
AbaqusArtifactPaths
```

Parameters

Name	Type	Required	Default	Description
`value`	`Union['IGAJob', 'CoreformIGAJobSpec']`	Yes	—	Portable Abaqus Job or its complete specification.

2 context.py

Module path: `coreform.job.context`

Available API members

Name	Kind	Description
`EnvironmentDelta.values`	Attribute	Environment variables assigned to exact values.
`EnvironmentDelta.prepends`	Attribute	Values prepended to path-like environment variables.
`ExecutionContext.plan`	Attribute	Ordered Job pipeline to execute or render.
`ExecutionContext.work_directory`	Attribute	Existing directory in which the pipeline operates.
`ExecutionContext.environment`	Attribute	Minimal environment changes required by the pipeline.

Name	Kind	Description
`EnvironmentDelta.apply`	Instance method	Apply this delta to a base environment and return the result.
`build_execution_context`	Function	Resolve a portable Job plan for one existing operation directory.
`EnvironmentDelta`	Class	The environment required by a Job without copying the parent environment.
`ExecutionContext`	Class	Bind an execution plan to a working directory and environment.

2.1 `EnvironmentDelta`

Object path: `coreform.job.context.EnvironmentDelta`

The environment required by a Job without copying the parent environment.

Call syntax

```
result = EnvironmentDelta()
```

Signatures

```
EnvironmentDelta(values: Tuple[Tuple[str, str], ...] = (), prepends:
Tuple[Tuple[str, str], ...] = ())
```

Parameters

Name	Type	Required	Default	Description
`values`	`Tuple[Tuple[str, str], ...]`	No	`()`	Environment variables assigned to exact values.
`prepends`	`Tuple[Tuple[str, str], ...]`	No	`()`	Values prepended to path-like envi-

Name	Type	Required	Default	Description
				ronment variables.

2.1.a Attributes

Name	Type	Value	Description
`values`	`Tuple[Tuple[str, str], ...]`	`()`	Environment variables assigned to exact values.
`prepends`	`Tuple[Tuple[str, str], ...]`	`()`	Values prepended to path-like environment variables.

2.1.b `apply`

Defined on: `coreform.job.context.EnvironmentDelta.apply`

Description: Apply this delta to a base environment and return the result.

Call syntax

```
result = environmentDelta.apply()
```

Returns

Type	Description
`dict[str, str]`	Apply this delta to a base environment and return the result.

Signatures

```
apply(base_environment: Optional[Mapping[str, str]] = None) -> dict[str, str]
```

Parameters

Name	Type	Required	Default	Description
`base_environment`	`Optional[Mapping[str, str]]`	No	`None`	Starting environment, or `None` to copy the current

Name	Type	Required	Default	Description
				process environment.

2.2 `ExecutionContext`

Object path: `coreform.job.context.ExecutionContext`

Bind an execution plan to a working directory and environment.

Call syntax

```
result = ExecutionContext(plan=plan, work_directory=work_directory, environment=environment)
```

Signatures

```
ExecutionContext(plan: ExecutionPlan, work_directory: str, environment: EnvironmentDelta)
```

Parameters

Name	Type	Required	Default	Description
`plan`	`ExecutionPlan`	Yes	—	Ordered Job pipeline to execute or render.
`work_directory`	`str`	Yes	—	Existing directory in which the pipeline operates.
`environment`	`EnvironmentDelta`	Yes	—	Minimal environment changes required by the pipeline.

2.2.a Attributes

Name	Type	Value	Description
`plan`	`ExecutionPlan`	—	Ordered Job pipeline to execute or render.
`work_directory`	`str`	—	Existing directory in which the pipeline operates.
`environment`	`EnvironmentDelta`	—	Minimal environment changes required by the pipeline.

2.3 `build_execution_context`

Object path: `coreform.job.context.build_execution_context`

Description: Resolve a portable Job plan for one existing operation directory.

Call syntax

```
result = coreform.job.context.build_execution_context(value=value, work_directory=work_directory)
```

Returns

Type	Description
`ExecutionContext`	Resolved execution plan, working directory, and environment changes.

Signatures

```
build_execution_context(value: Union['IGAJob', 'CoreformIGAJobSpec'], *, work_directory: str, base_environment: Optional[Mapping[str, str]] = None) -> ExecutionContext
```

Parameters

Name	Type	Required	Default	Description
`value`	`Union['IGAJob', 'CoreformIGAJobSpec']`	Yes	—	Portable Job or complete

Name	Type	Required	Default	Description
				Job specification.
`work_directory`	`str`	Yes	—	Existing directory containing the Job's input artifacts.
`base_environment`	`Optional[Mapping[str, str]]`	No	`None`	Environment used to resolve executables and path additions.

3 export.py

Module path: `coreform.job.export`

Available API members

Name	Kind	Description
`SCRIPT_DIALECTS`	Attribute	Canonical names accepted by `render_script`.
`to_bash`	Function	Render a Job or execution plan as a runnable local bash script.
`to_zsh`	Function	Render a Job or execution plan as a runnable local zsh script.
`to_powershell`	Function	Render a Job or execution plan as a runnable local PowerShell script.
`to_cmd`	Function	Render a Job or execution plan as a runnable Windows CMD batch script.
`to_slurm`	Function	Render a SLURM batch-script starting point.
`to_pbs`	Function	Render a PBS/OpenPBS batch-script starting point.

Name	Kind	Description
`to_batch_script`	Function	Render the batch-script dialect selected by the Job's remote queue.
`render_script`	Function	Render one explicitly selected script dialect.
`ExportError`	Class	Report that an execution plan cannot be rendered in a requested dialect.

3.1 `ExportError`

Object path: `coreform.job.export.ExportError`

Report that an execution plan cannot be rendered in a requested dialect.

3.2 `SCRIPT_DIALECTS`

Object path: `coreform.job.export.SCRIPT_DIALECTS`

Type: `Tuple[str, ...]`

Canonical names accepted by `render_script`.

3.3 `to_bash`

Object path: `coreform.job.export.to_bash`

Description: Render a Job or execution plan as a runnable local bash script.

Call syntax

```
result = coreform.job.export.to_bash(plan_or_job=plan_or_job)
```

Returns

Type	Description
`str`	Render a Job or execution plan as a runnable local bash script.

Signatures

```
to_bash(plan_or_job: Union['IGAJob', 'CoreformIGAJobSpec', ExecutionPlan, ExecutionContext], *, script_directory: bool = False) -> str
```

Parameters

Name	Type	Required	Default	Description
`plan_or_job`	`Union['IGAJob', 'CoreformIGAJobSpec', ExecutionPlan, ExecutionContext]`	Yes	—	Job definition, projected plan, or resolved execution context to render.
`script_directory`	`bool`	No	`False`	Run from the script's directory instead of the context working directory.

3.4 `to_zsh`

Object path: `coreform.job.export.to_zsh`

Description: Render a Job or execution plan as a runnable local zsh script.

Call syntax

```
result = coreform.job.export.to_zsh(plan_or_job=plan_or_job)
```

Returns

Type	Description
`str`	Render a Job or execution plan as a runnable local zsh script.

Signatures

```
to_zsh(plan_or_job: Union['IGAJob', 'CoreformIGAJobSpec', ExecutionPlan, ExecutionContext], *, script_directory: bool = False) -> str
```

Parameters

Name	Type	Required	Default	Description
`plan_or_job`	`Union['IGAJob', 'CoreformIGAJobSpec', ExecutionPlan, ExecutionContext]`	Yes	—	Job definition, projected plan, or resolved execution context to render.

Name	Type	Required	Default	Description
`script_directory`	`bool`	No	`False`	Run from the script's directory instead of the context working directory.

3.5 `to_powershell`

Object path: `coreform.job.export.to_powershell`

Description: Render a Job or execution plan as a runnable local PowerShell script.

Call syntax

```
result = coreform.job.export.to_powershell(plan_or_job=plan_or_job)
```

Returns

Type	Description
`str`	Render a Job or execution plan as a runnable local PowerShell script.

Signatures

```
to_powershell(plan_or_job: Union['IGAJob', 'CoreformIGAJobSpec', ExecutionPlan,
ExecutionContext], *, script_directory: bool = False) -> str
```

Parameters

Name	Type	Required	Default	Description
`plan_or_job`	`Union['IGAJob', 'CoreformIGAJobSpec', ExecutionPlan, ExecutionContext]`	Yes	—	Job definition, projected plan, or resolved execution context to render.
`script_directory`	`bool`	No	`False`	Run from the script's directory instead of the context

Name	Type	Required	Default	Description
				working directory.

3.6 `to_cmd`

Object path: `coreform.job.export.to_cmd`

Description: Render a Job or execution plan as a runnable Windows CMD batch script.

Call syntax

```
result = coreform.job.export.to_cmd(plan_or_job=plan_or_job)
```

Returns

Type	Description
`str`	Render a Job or execution plan as a runnable Windows CMD batch script.

Signatures

```
to_cmd(plan_or_job: Union['IGAJob', 'CoreformIGAJobSpec', ExecutionPlan, ExecutionContext], *, script_directory: bool = False) -> str
```

Parameters

Name	Type	Required	Default	Description
`plan_or_job`	`Union['IGAJob', 'CoreformIGAJobSpec', ExecutionPlan, ExecutionContext]`	Yes	—	Job definition, projected plan, or resolved execution context to render.
`script_directory`	`bool`	No	`False`	Run from the script's directory instead of the context working directory.

3.7 `to_slurm`

Object path: `coreform.job.export.to_slurm`

Description: Render a SLURM batch-script starting point.

Call syntax

```
result = coreform.job.export.to_slurm(plan_or_job=plan_or_job)
```

Returns

Type	Description
`str`	Render a SLURM batch-script starting point.

Signatures

```
to_slurm(plan_or_job: Union['IGAJob', 'CoreformIGAJobSpec', ExecutionPlan,
ExecutionContext], *, script_directory: bool = False) -> str
```

Parameters

Name	Type	Required	Default	Description
`plan_or_job`	`Union['IGAJob', 'CoreformIGAJobSpec', ExecutionPlan, ExecutionContext]`	Yes	—	Job definition, projected plan, or resolved execution context to render.
`script_directory`	`bool`	No	`False`	Run from the script's directory instead of the context working directory.

3.8 `to_pbs`

Object path: `coreform.job.export.to_pbs`

Description: Render a PBS/OpenPBS batch-script starting point.

Call syntax

```
result = coreform.job.export.to_pbs(plan_or_job=plan_or_job)
```

Returns

Type	Description
`str`	Render a PBS/OpenPBS batch-script starting point.

Signatures

```
to_pbs(plan_or_job:  Union['IGAJob',  'CoreformIGAJobSpec',  ExecutionPlan,
ExecutionContext], *, script_directory: bool = False) -> str
```

Parameters

Name	Type	Required	Default	Description
`plan_or_job`	`Union['IGAJob', 'CoreformIGAJobSpec', ExecutionPlan, ExecutionContext]`	Yes	—	Job definition, projected plan, or resolved execution context to render.
`script_directory`	`bool`	No	`False`	Run from the script's directory instead of the context working directory.

3.9 `to_batch_script`

Object path: `coreform.job.export.to_batch_script`

Description: Render the batch-script dialect selected by the Job's remote queue.

Call syntax

```
result = coreform.job.export.to_batch_script(plan_or_job=plan_or_job)
```

Returns

Type	Description
`str`	Render the batch-script dialect selected by the Job's remote queue.

Signatures

```
to_batch_script(plan_or_job: Union['IGAJob', 'CoreformIGAJobSpec',
ExecutionPlan, ExecutionContext], *, script_directory: bool = False) -> str
```

Parameters

Name	Type	Required	Default	Description
`plan_or_job`	`Union['IGAJob', 'CoreformIGAJobSpec', ExecutionPlan, ExecutionContext]`	Yes	—	Job definition, projected plan, or resolved execution context to render.
`script_directory`	`bool`	No	`False`	Run from the script's directory instead of the context working directory.

Raises

Type	Condition
`ExportError`	If no supported SLURM or PBS remote queue selects the dialect.

3.10 `render_script`

Object path: `coreform.job.export.render_script`

Description: Render one explicitly selected script dialect.

Call syntax

```
result = coreform.job.export.render_script(plan_or_job=plan_or_job,
dialect=dialect)
```

Returns

Type	Description
`str`	Complete script text in the selected dialect.

Signatures

```
render_script(plan_or_job: Union['IGAJob', 'CoreformIGAJobSpec', ExecutionPlan,
ExecutionContext], dialect: str, *, script_directory: bool = False) -> str
```

Parameters

Name	Type	Required	Default	Description
`plan_or_job`	`Union['IGAJob', 'CoreformIGAJobSpec', ExecutionPlan, ExecutionContext]`	Yes	—	Job definition, projected plan, or resolved execution context to render.
`dialect`	`str`	Yes	—	One of the uppercase names in `SCRIPT_DIALECTS`.
`script_directory`	`bool`	No	`False`	Run from the script's directory instead of the context working directory.

Raises

Type	Condition
`ExportError`	If `dialect` is not supported.

4 model.py

Module path: `coreform.job.model`

Available API members

Name	Kind	Description
`IGAJob.spec`	Attribute	Complete portable Job specification to wrap.
`IGAJob.name`	Attribute	Return the Job name.
`IGAJob.model`	Attribute	Return the Abaqus model name, or `None` for a native target.

Name	Kind	Description
`IGAJob.create`	Class method	Generic entry: one positional `CoreformIGAJobSpec` (import path) or thenested keyword form (`name=`, `mesh=MeshStage(...)`,`solv` For the flat Abaqus ergonomics useIGAJob.abaqus(...).
`IGAJob.abaqus`	Class method	Flat ergonomic assembler for the common Abaqus job: project flat keywordsinto the nested stage-pipeline spec. `num_iga_cpus` is the mesh stage's cpu-count (`num_cpus` is the solve stage's). `queue=` fills every stage; aper-stage `mesh_queue=` / `interop_queue=` / `solve_queue=` overrides-one. This is the portable analogue of the flat public `Job(...)` adapter-constructor; less common stage inputs (parts, dedup_epsilon, ...) use thenested spec form.
`IGAJob.from_spec`	Class method	Wrap an existing portable Job specification.
`IGAJob.to_spec`	Instance method	Return the wrapped portable Job specification.
`IGAJob.copy_with`	Instance method	Return a validated Job copy with selected fields replaced.
`IGAJob`	Class	Wrap one portable Coreform IGA Job specification.

4.1 `IGAJob`

Object path: `coreform.job.model.IGAJob`

Wrap one portable Coreform IGA Job specification.

Call syntax

```
result = IGAJob(spec=spec)
```

Signatures

```
IGAJob(spec: CoreformIGAJobSpec)
```

Parameters

Name	Type	Required	Default	Description
`spec`	`CoreformIGAJobSpec`	Yes	—	Complete portable Job specification to wrap.

Example

```
>>> job = IGAJob.abaqus(name="Job-1", model="Model-1")
```

4.1.a Attributes

Name	Type	Value	Description
`spec`	`CoreformIGAJobSpec`	—	Complete portable Job specification to wrap.
`name`	`str`	—	Return the Job name.
`model`	`Optional[str]`	—	Return the Abaqus model name, or `None` for a native target.

4.1.b `create`

Defined on: `coreform.job.model.IGAJob.create`

Description: Generic entry: one positional `CoreformIGAJobSpec` (import path) or the nested keyword form (`name=`, `mesh=MeshStage(...)`, `solver=AbaqusTarget(...)`). For the flat Abaqus ergonomics use `IGAJob.abaqus(...)`.

Call syntax

```
result = IGAJob.create(spec)
result = IGAJob.create(name=name, mesh=mesh, solver=solver)
```

Returns

Type	Description
`IGAJob`	Generic entry: one positional `CoreformIGAJobSpec` (import path) or thenested keyword form (`name=`, `mesh=MeshStage(...)`,`solver=Abaqu For the flat Abaqus ergonomics use`IGAJob.abaqus(...)`.

Signatures

```
create(spec: CoreformIGAJobSpec, /) -> IGAJob
create(name: str = None, description: str = '', mesh: Optional[MeshStage] = None,
solver: Optional[Union[AbaqusTarget, NativeTarget]] = None) -> IGAJob
```

Parameters

For `create(spec: CoreformIGAJobSpec, /) -> IGAJob`:

Name	Type	Required	Default	Description
`spec`	`CoreformIGAJobSpe`	Yes	—	Complete portable specification.

For `create(name: str = None, description: str = '', mesh: Optional[MeshStage] = None, solver: Optional[Union[AbaqusTarget, NativeTarget]] = None) -> IGAJob`:

Name	Type	Required	Default	Description
`name`	`str`	Yes	—	Unique, non-empty Job name.
`description`	`str`	No	`''`	Optional human-readable

Name	Type	Required	Default	Description
				Job description.
`mesh`	`Optional[MeshStage]`	Yes	—	Required Coreform IGA Mesh stage.
`solver`	`Optional[Union[AbaqusTarget, NativeTarget]]`	Yes	—	Required downstream solver target.

4.1.c `abaqus`

Defined on: `coreform.job.model.IGAJob.abaqus`

Description: Flat ergonomic assembler for the common Abaqus job: project flat keywords into the nested stage-pipeline spec. `num_iga_cpus` is the mesh stage's cpu count (`num_cpus` is the solve stage's). `queue=` fills every stage; a per-stage `mesh_queue=` / `interop_queue=` / `solve_queue=` overrides one. This is the portable analogue of the flat public `Job(...)` adapter constructor; less common stage inputs (parts, dedup_epsilon, ...) use the nested spec form.

Call syntax

```
result = IGAJob.abaqus()
```

Returns

Type	Description
`IGAJob`	Flat ergonomic assembler for the common Abaqus job: project flat keywordsinto the nested stage-pipeline spec. `num_iga_cpus` is the mesh stage's cpucount (`num_cpus` is the solve stage's). `queue=` fills every stage; aper-stage `mesh_queue=` / `interop_queue=` / `solve_queue=` overridesone. This is the portable analogue of the flat public `Job(...)` adapterconstructor; less common stage inputs (parts, dedup_epsilon, ...) use thenested spec form.

Signatures

```
abaqus(name: Optional[str] = None, model: Optional[str] = None, description:
str = '', mesh_executable: str = '', mpiexec: str = '', num_iga_cpus:
int = 1, output_db: Optional[str] = None, interop_executable: str =
'', inp_prefix: Optional[str] = None, num_cpus: int = 1, num_gpus:
int = 0, parallel_mode: Union[ParallelMode, str] = ParallelMode.THREADS,
threads_per_process: Optional[int] = None, num_domains: Optional[int] = None,
memory: int = 90, memory_units: Union[MemoryUnits, str] = MemoryUnits.PERCENTAGE,
queue: Optional[Union[Local, RemoteQueue, Mapping[str, Any]]] = None,
mesh_queue: Optional[Union[Local, RemoteQueue, Mapping[str, Any]]] = None,
interop_queue: Optional[Union[Local, RemoteQueue, Mapping[str, Any]]] = None,
solve_queue: Optional[Union[Local, RemoteQueue, Mapping[str, Any]]] = None) -
> 'IGAJob'
```

Parameters

Name	Type	Required	Default	Description
`name`	`Optional[str]`	No	`None`	Unique Job name.
`model`	`Optional[str]`	No	`None`	Abaqus model used by the solve pipeline.
`description`	`str`	No	`''`	Optional human-readable Job description.
`mesh_executable`	`str`	No	`''`	Coreform IGA Mesh executable override.
`mpiexec`	`str`	No	`''`	MPI launcher override for the Mesh stage.
`num_iga_cpus`	`int`	No	`1`	CPU count for the Mesh stage.
`output_db`	`Optional[str]`	No	`None`	Mesh database filename override.

Name	Type	Required	Default	Description
`interop_executable`	`str`	No	`''`	Abaqus interop executable override.
`inp_prefix`	`Optional[str]`	No	`None`	Basename override for the Abaqus artifact family.
`num_cpus`	`int`	No	`1`	CPU count for the Abaqus solve stage.
`num_gpus`	`int`	No	`0`	GPU count for the Abaqus solve stage.
`parallel_mode`	`Union[ParallelMode, str]`	No	`ParallelMode.THREADS`	Abaqus process and thread mode.
`threads_per_process`	`Optional[int]`	No	`None`	Threads assigned to each Abaqus MPI process.
`num_domains`	`Optional[int]`	No	`None`	Abaqus solver-domain count.
`memory`	`int`	No	`90`	Abaqus memory limit interpreted using `memory_units`.
`memory_units`	`Union[MemoryUnits, str]`	No	`MemoryUnits.PERCENTAGE`	Units used to interpret `memory`.
`queue`	`Optional[Union[Local, RemoteQueue, Mapping[str, Any]]]`	No	`None`	Default submission settings for all stages.

Name	Type	Required	Default	Description
`mesh_queue`	`Optional[Union[Local, RemoteQueue, Mapping[str, Any]]]`	No	`None`	Mesh-stage submission override.
`interop_queue`	`Optional[Union[Local, RemoteQueue, Mapping[str, Any]]]`	No	`None`	Interop-stage submission override.
`solve_queue`	`Optional[Union[Local, RemoteQueue, Mapping[str, Any]]]`	No	`None`	Solve-stage submission override.

4.1.d `from_spec`

Defined on: `coreform.job.model.IGAJob.from_spec`

Description: Wrap an existing portable Job specification.

Call syntax

```
result = IGAJob.from_spec(spec=spec)
```

Returns

Type	Description
`IGAJob`	Wrap an existing portable Job specification.

Signatures

```
from_spec(spec: CoreformIGAJobSpec) -> 'IGAJob'
```

Parameters

Name	Type	Required	Default	Description
`spec`	`CoreformIGAJobSpec`	Yes	—	Complete portable Job specification to wrap.

4.1.e `to_spec`

Defined on: `coreform.job.model.IGAJob.to_spec`

Description: Return the wrapped portable Job specification.

Call syntax

```
result = job.to_spec()
```

Returns

Type	Description
`CoreformIGAJobSpec`	Return the wrapped portable Job specification.

Signatures

```
to_spec() -> CoreformIGAJobSpec
```

Parameters

None.

4.1.f `copy_with`

Defined on: `coreform.job.model.IGAJob.copy_with`

Description: Return a validated Job copy with selected fields replaced.

Call syntax

```
result = job.copy_with(...)
```

Returns

Type	Description
`IGAJob`	Return a validated Job copy with selected fields replaced.

Signatures

```
copy_with(*, name: str = ..., description: str = ..., mesh: Optional[MeshStage]
= ..., solver: Optional[Union[AbaqusTarget, NativeTarget]] = ...) -> IGAJob
```

Parameters

Name	Type	Required	Default	Description
`name`	`str`	No	Current value	Unique, non-empty Job name.
`description`	`str`	No	Current value	Optional human-readable Job description.
`mesh`	`Optional[MeshStage]`	No	Current value	Required Coreform IGA Mesh stage.
`solver`	`Optional[Union[AbaqusTarget, NativeTarget]]`	No	Current value	Required downstream solver target.

5 plan.py

Module path: `coreform.job.plan`

Available API members

Name	Kind	Description
`Command.argv`	Attribute	Executable and arguments in process-launch order.
`RestoreFirst.protected_copy`	Attribute	Backup input copied into place before a guarded rerun.
`RestoreFirst.destination`	Attribute	Input path restored when it contains the translated marker.
`RestoreFirst.translated_marker`	Attribute	Text identifying a previously translated destination.
`PlanStep.name`	Attribute	Stable stage name used by dependencies and results.
`PlanStep.command`	Attribute	Executable invocation for this stage.

Name	Kind	Description
`PlanStep.submission`	Attribute	Requested local or scheduler execution policy.
`PlanStep.inputs`	Attribute	Artifact paths consumed by this stage.
`PlanStep.outputs`	Attribute	Artifact paths produced by this stage.
`PlanStep.depends_on`	Attribute	Stage names that must finish before this stage runs.
`PlanStep.restore_first`	Attribute	Optional rerun guard applied before the command.
`ExecutionPlan.job_name`	Attribute	Logical Job name associated with the pipeline.
`ExecutionPlan.steps`	Attribute	Pipeline stages in deterministic execution order.
`build_execution_plan`	Function	Project a job (IGAJob or CoreformIGAJobSpec) into an ordered ExecutionPlan.
`PlanError`	Class	Report that a portable Job cannot be projected into an execution plan.
`Command`	Class	A single executable invocation.
`RestoreFirst`	Class	Rerun guard for a non-idempotent step: before running, if `destination`already contains `translated_marker` (i.e. it is the step's own prior OUTPUT,not the raw input) and `protected_copy` exists, copy protected_copy back overdestination.
`PlanStep`	Class	One pipeline stage as a runnable step: its com-

Name	Kind	Description
		mand, where it runs, theartifacts it reads/writes, the steps it must follow, and an optional rerun-guard (see RestoreFirst).
`ExecutionPlan`	Class	Describe the ordered, backend-neutral stages for one Job.

5.1 `PlanError`

Object path: `coreform.job.plan.PlanError`

Report that a portable Job cannot be projected into an execution plan.

5.2 `Command`

Object path: `coreform.job.plan.Command`

A single executable invocation.

Call syntax

```
result = Command(argv=argv)
```

Signatures

```
Command(argv: Tuple[str, ...])
```

Parameters

Name	Type	Required	Default	Description
`argv`	`Tuple[str, ...]`	Yes	—	Executable and arguments in process-launch order.

5.2.a Attributes

Name	Type	Value	Description
`argv`	`Tuple[str, ...]`	—	Executable and arguments in

Name	Type	Value	Description
			process-launch order.

5.3 `RestoreFirst`

Object path: `coreform.job.plan.RestoreFirst`

Rerun guard for a non-idempotent step: before running, if `destination` already contains `translated_marker` (i.e. it is the step's own prior OUTPUT, not the raw input) and `protected_copy` exists, copy protected_copy back over destination.

Call syntax

```
result = RestoreFirst(protected_copy=protected_copy, destination=destination, translated_marker=translated_marker)
```

Signatures

```
RestoreFirst(protected_copy: str, destination: str, translated_marker: str)
```

Parameters

Name	Type	Required	Default	Description
`protected_copy`	`str`	Yes	—	Backup input copied into place before a guarded rerun.
`destination`	`str`	Yes	—	Input path restored when it contains the translated marker.
`translated_marker`	`str`	Yes	—	Text identifying a previously translated destination.

5.3.a Attributes

Name	Type	Value	Description
`protected_copy`	`str`	—	Backup input copied into place before a guarded rerun.
`destination`	`str`	—	Input path restored when it contains the translated marker.
`translated_marker`	`str`	—	Text identifying a previously translated destination.

5.4 `PlanStep`

Object path: `coreform.job.plan.PlanStep`

One pipeline stage as a runnable step: its command, where it runs, the artifacts it reads/writes, the steps it must follow, and an optional rerun guard (see RestoreFirst).

Call syntax

```
result = PlanStep(name=name, command=command, submission=submission, inputs=inputs, outputs=outputs, depends_on=depends_on)
```

Signatures

```
PlanStep(name: str, command: Command, submission: Optional[Union[Local, RemoteQueue]], inputs: Tuple[str, ...], outputs: Tuple[str, ...], depends_on: Tuple[str, ...], restore_first: Optional[RestoreFirst] = None)
```

Parameters

Name	Type	Required	Default	Description
`name`	`str`	Yes	—	Stable stage name used by dependencies and results.

Name	Type	Required	Default	Description
`command`	`Command`	Yes	—	Executable invocation for this stage.
`submission`	`Optional[Union[Local, RemoteQueue]]`	Yes	—	Requested local or scheduler execution policy.
`inputs`	`Tuple[str, ...]`	Yes	—	Artifact paths consumed by this stage.
`outputs`	`Tuple[str, ...]`	Yes	—	Artifact paths produced by this stage.
`depends_on`	`Tuple[str, ...]`	Yes	—	Stage names that must finish before this stage runs.
`restore_first`	`Optional[RestoreFirst]`	No	`None`	Optional rerun guard applied before the command.

5.4.a Attributes

Name	Type	Value	Description
`name`	`str`	—	Stable stage name used by dependencies and results.
`command`	`Command`	—	Executable invocation for this stage.
`submission`	`Optional[Union[Local, RemoteQueue]]`	—	Requested local or scheduler execution policy.
`inputs`	`Tuple[str, ...]`	—	Artifact paths consumed by this stage.

Name	Type	Value	Description
`outputs`	`Tuple[str, ...]`	—	Artifact paths produced by this stage.
`depends_on`	`Tuple[str, ...]`	—	Stage names that must finish before this stage runs.
`restore_first`	`Optional[RestoreFirst]`	`None`	Optional rerun guard applied before the command.

5.5 `ExecutionPlan`

Object path: `coreform.job.plan.ExecutionPlan`

Describe the ordered, backend-neutral stages for one Job.

Call syntax

```
result = ExecutionPlan(job_name=job_name, steps=steps)
```

Signatures

```
ExecutionPlan(job_name: str, steps: Tuple[PlanStep, ...])
```

Parameters

Name	Type	Required	Default	Description
`job_name`	`str`	Yes	—	Logical Job name associated with the pipeline.
`steps`	`Tuple[PlanStep, ...]`	Yes	—	Pipeline stages in deterministic execution order.

Example

```
>>> plan = ExecutionPlan(
...     job_name="Job-1",
...     steps=(PlanStep(
```

```
...         name="mesh",
...         command=Command(("coreform_iga_mesh", "model.cf")),
...         submission=None,
...         inputs=("model.cf",),
...         outputs=("model.sql",),
...         depends_on=(),
...     ),),
... )
```

5.5.a Attributes

Name	Type	Value	Description
`job_name`	`str`	—	Logical Job name associated with the pipeline.
`steps`	`Tuple[PlanStep, ...]`	—	Pipeline stages in deterministic execution order.

5.6 `build_execution_plan`

Object path: `coreform.job.plan.build_execution_plan`

Description: Project a job (IGAJob or CoreformIGAJobSpec) into an ordered ExecutionPlan.

Call syntax

```
result = coreform.job.plan.build_execution_plan(job=job)
```

Returns

Type	Description
`ExecutionPlan`	Ordered, backend-neutral Mesh, interop, and solve stages.

Signatures

```
build_execution_plan(job: Union['IGAJob', CoreformIGAJobSpec]) -> ExecutionPlan
```

Parameters

Name	Type	Required	Default	Description
`job`	`Union['IGAJob', CoreformIGAJobSpec]`	Yes	—	Portable Job or complete Job specification to project.

Raises

Type	Condition
`PlanError`	If the Job uses a native or otherwise unsupported solver target.

6 results.py

Module path: `coreform.job.results`

Available API members

Name	Kind	Description
`JobStepResult.name`	Attribute	Stable pipeline-stage name.
`JobStepResult.returncode`	Attribute	Process exit code returned by the completed stage.
`JobStepResult.ok`	Attribute	Report whether the stage completed successfully.
`JobExecutionResult.results`	Attribute	Results in pipeline execution order.
`JobExecutionResult.ok`	Attribute	Report whether every completed stage succeeded.
`JobExecutionResult.get`	Instance method	Return the result for a named stage, or `None` when it did not run.
`JobStepResult`	Class	Report the exit status of one completed Job pipeline stage.

Name	Kind	Description
`JobExecutionResult`	Class	Report the ordered stage results from a Job execution operation.

6.1 `JobStepResult`

Object path: `coreform.job.results.JobStepResult`

Report the exit status of one completed Job pipeline stage.

Call syntax

```
result = JobStepResult(name=name, returncode=returncode)
```

Signatures

```
JobStepResult(name: str, returncode: int)
```

Parameters

Name	Type	Required	Default	Description
`name`	`str`	Yes	—	Stable pipeline-stage name.
`returncode`	`int`	Yes	—	Process exit code returned by the completed stage.

6.1.a Attributes

Name	Type	Value	Description
`name`	`str`	—	Stable pipeline-stage name.
`returncode`	`int`	—	Process exit code returned by the completed stage.
`ok`	`bool`	—	Report whether the stage completed successfully.

6.2 `JobExecutionResult`

Object path: `coreform.job.results.JobExecutionResult`

Report the ordered stage results from a Job execution operation.

Call syntax

```
result = JobExecutionResult(results=results)
```

Signatures

```
JobExecutionResult(results: Tuple[JobStepResult, ...])
```

Parameters

Name	Type	Required	Default	Description
`results`	`Tuple[JobStepResult, ...]`	Yes	—	Results in pipeline execution order.

6.2.a Attributes

Name	Type	Value	Description
`results`	`Tuple[JobStepResult, ...]`	—	Results in pipeline execution order.
`ok`	`bool`	—	Report whether every completed stage succeeded.

6.2.b `get`

Defined on: `coreform.job.results.JobExecutionResult.get`

Description: Return the result for a named stage, or `None` when it did not run.

Call syntax

```
result = jobExecutionResult.get(name=name)
```

Returns

Type	Description
`JobStepResult | None`	Return the result for a named stage, or `None` when it did not run.

Signatures

```
get(name: str) -> JobStepResult | None
```

Parameters

Name	Type	Required	Default	Description
`name`	`str`	Yes	—	Pipeline-stage name to find.

7 specs.py

Module path: `coreform.job.specs`

Available API members

Name	Kind	Description
`MemoryUnits.PERCENTAGE`	Attribute	Percentage of the physical memory available to Abaqus.
`MemoryUnits.MEGA_BYTES`	Attribute	Memory limit expressed in megabytes.
`MemoryUnits.GIGA_BYTES`	Attribute	Memory limit expressed in gigabytes.
`ParallelMode.DEFAULT`	Attribute	Let Abaqus select its default process and thread layout.
`ParallelMode.MPI`	Attribute	Use one thread per MPI process.
`ParallelMode.THREADS`	Attribute	Use a single process with multiple threads.
`ParallelMode.HYBRID`	Attribute	Combine MPI processes with multiple threads per process.

Name	Kind	Description
`Scheduler.ABAQUS`	Attribute	Delegate queue submission to the Abaqus site environment.
`Scheduler.SLURM`	Attribute	Render SLURM scheduler directives.
`Scheduler.PBS`	Attribute	Render PBS/OpenPBS scheduler directives.
`Scheduler.LSF`	Attribute	Describe an LSF queue for downstream handling.
`Local.schemaVersion`	Attribute	Version of the serialized specification schema.
`Local.submission_type`	Attribute	Serialized discriminator for local execution.
`RemoteQueue.schemaVersion`	Attribute	Version of the serialized specification schema.
`RemoteQueue.scheduler`	Attribute	Scheduler responsible for the remote allocation.
`RemoteQueue.queue_name`	Attribute	Site queue or partition name.
`RemoteQueue.account`	Attribute	Scheduler account or project identifier.
`RemoteQueue.walltime`	Attribute	Scheduler wall-time limit.
`RemoteQueue.nodes`	Attribute	Requested node count.
`RemoteQueue.extra_directives`	Attribute	Additional site-specific scheduler directive lines.
`RemoteQueue.submission_type`	Attribute	Serialized discriminator for remote-queue execution.
`MeshStage.schemaVersion`	Attribute	Version of the serialized specification schema.
`MeshStage.mesh_executable`	Attribute	Coreform IGA Mesh executable name or path override.

Name	Kind	Description
`MeshStage.mpiexec`	Attribute	MPI launcher name or path used when `num_cpus` is greater than one.
`MeshStage.num_cpus`	Attribute	Positive CPU count for Mesh generation.
`MeshStage.cf_file`	Attribute	Coreform input filename, or `None` to derive it from the Job.
`MeshStage.output_db`	Attribute	Mesh database filename, or `None` to derive it from the input.
`MeshStage.parts`	Attribute	Part names to mesh, or an empty tuple for all parts.
`MeshStage.extra_args`	Attribute	Additional command-line arguments passed to the Mesh executable.
`MeshStage.queue`	Attribute	Submission policy for the Mesh stage.
`AbaqusInterop.schemaVersion`	Attribute	Version of the serialized specification schema.
`AbaqusInterop.interop_executable`	Attribute	Coreform Abaqus interop executable name or path override.
`AbaqusInterop.cf_file`	Attribute	Coreform input filename, or `None` to use the Mesh-stage input.
`AbaqusInterop.sql_file`	Attribute	Mesh database filename, or `None` to use the Mesh-stage output.
`AbaqusInterop.inp_prefix`	Attribute	Basename for the Abaqus input and result artifact family.
`AbaqusInterop.parts`	Attribute	Part names to translate, or an empty tuple for all parts.

Name	Kind	Description
`AbaqusInterop.dedup_epsilon`	Attribute	Optional nonnegative node-deduplication tolerance.
`AbaqusInterop.keyword_report`	Attribute	Keyword-report filename, or `None` to derive it from the prefix.
`AbaqusInterop.extra_args`	Attribute	Additional arguments passed to the interop executable.
`AbaqusInterop.queue`	Attribute	Submission policy for the interop stage.
`AbaqusSolve.schemaVersion`	Attribute	Version of the serialized specification schema.
`AbaqusSolve.num_cpus`	Attribute	Positive CPU count for the Abaqus solve.
`AbaqusSolve.num_gpus`	Attribute	Nonnegative GPU count for the Abaqus solve.
`AbaqusSolve.parallel_mode`	Attribute	Solver process and thread mode.
`AbaqusSolve.threads_per_process`	Attribute	Thread count per MPI process for an explicit hybrid layout.
`AbaqusSolve.num_domains`	Attribute	Abaqus solver-domain count; it must be a multiple of `num_cpus`.
`AbaqusSolve.memory`	Attribute	Positive solver memory limit.
`AbaqusSolve.memory_units`	Attribute	Units used to interpret `memory`.
`AbaqusSolve.queue`	Attribute	Submission policy for the solve stage.
`NativeSolve.schemaVersion`	Attribute	Version of the serialized specification schema.

Name	Kind	Description
`NativeSolve.num_cpus`	Attribute	Positive CPU count for the native solve.
`NativeSolve.queue`	Attribute	Submission policy for the native solve stage.
`AbaqusTarget.schemaVersion`	Attribute	Version of the serialized specification schema.
`AbaqusTarget.model`	Attribute	Nonempty Abaqus model name.
`AbaqusTarget.interop`	Attribute	Required Coreform-to-Abaqus interop stage.
`AbaqusTarget.solve`	Attribute	Required Abaqus solve stage.
`AbaqusTarget.solver_type`	Attribute	Serialized discriminator for an Abaqus target.
`NativeTarget.schemaVersion`	Attribute	Version of the serialized specification schema.
`NativeTarget.solve`	Attribute	Required native Coreform IGA solve stage.
`NativeTarget.solver_type`	Attribute	Serialized discriminator for a native target.
`CoreformIGAJobSpec.schemaVersion`	Attribute	Version of the serialized specification schema.
`CoreformIGAJobSpec.name`	Attribute	Unique, nonempty Job name.
`CoreformIGAJobSpec.description`	Attribute	Optional human-readable Job description.
`CoreformIGAJobSpec.mesh`	Attribute	Required Coreform IGA Mesh stage.
`CoreformIGAJobSpec.solver`	Attribute	Required downstream solver target.
`Local.validate`	Instance method	Return validation issues for this specification.

Name	Kind	Description
`Local.to_dict`	Instance method	Serialize this specification to a versioned dictionary.
`Local.to_json`	Instance method	Serialize this specification to a versioned JSON document.
`Local.from_dict`	Class method	Create a validated submission specification from a mapping payload.
`Local.from_json`	Class method	Create a validated specification from a JSON document.
`Local.copy_with`	Instance method	Return a validated copy with the supplied fields replaced.
`RemoteQueue.to_dict`	Instance method	Serialize this specification to a versioned dictionary.
`RemoteQueue.to_json`	Instance method	Serialize this specification to a versioned JSON document.
`RemoteQueue.from_dict`	Class method	Create a validated submission specification from a mapping payload.
`RemoteQueue.from_json`	Class method	Create a validated specification from a JSON document.
`RemoteQueue.copy_with`	Instance method	Return a validated copy with the supplied fields replaced.
`RemoteQueue.validate`	Instance method	Return validation issues for the remote queue settings.
`MeshStage.to_dict`	Instance method	Serialize this specification to a versioned dictionary.
`MeshStage.to_json`	Instance method	Serialize this specification to a versioned JSON document.

Name	Kind	Description
`MeshStage.from_dict`	Class method	Create a validated specification from a mapping payload.
`MeshStage.from_json`	Class method	Create a validated specification from a JSON document.
`MeshStage.copy_with`	Instance method	Return a validated copy with the supplied fields replaced.
`MeshStage.validate`	Instance method	Return validation issues for the Mesh stage.
`AbaqusInterop.to_dict`	Instance method	Serialize this specification to a versioned dictionary.
`AbaqusInterop.to_json`	Instance method	Serialize this specification to a versioned JSON document.
`AbaqusInterop.from_dict`	Class method	Create a validated specification from a mapping payload.
`AbaqusInterop.from_json`	Class method	Create a validated specification from a JSON document.
`AbaqusInterop.copy_with`	Instance method	Return a validated copy with the supplied fields replaced.
`AbaqusInterop.validate`	Instance method	Return validation issues for the Abaqus interop stage.
`AbaqusSolve.to_dict`	Instance method	Serialize this specification to a versioned dictionary.
`AbaqusSolve.to_json`	Instance method	Serialize this specification to a versioned JSON document.
`AbaqusSolve.from_dict`	Class method	Create a validated specification from a mapping payload.

Name	Kind	Description
`AbaqusSolve.from_json`	Class method	Create a validated specification from a JSON document.
`AbaqusSolve.copy_with`	Instance method	Return a validated copy with the supplied fields replaced.
`AbaqusSolve.validate`	Instance method	Return validation issues for the Abaqus solve settings.
`NativeSolve.to_dict`	Instance method	Serialize this specification to a versioned dictionary.
`NativeSolve.to_json`	Instance method	Serialize this specification to a versioned JSON document.
`NativeSolve.from_dict`	Class method	Create a validated specification from a mapping payload.
`NativeSolve.from_json`	Class method	Create a validated specification from a JSON document.
`NativeSolve.copy_with`	Instance method	Return a validated copy with the supplied fields replaced.
`NativeSolve.validate`	Instance method	Return validation issues for the native solve settings.
`AbaqusTarget.to_dict`	Instance method	Serialize this specification to a versioned dictionary.
`AbaqusTarget.to_json`	Instance method	Serialize this specification to a versioned JSON document.
`AbaqusTarget.from_dict`	Class method	Create a validated solver target from a mapping payload.
`AbaqusTarget.from_json`	Class method	Create a validated specification from a JSON document.

Name	Kind	Description
`AbaqusTarget.copy_with`	Instance method	Return a validated copy with the supplied fields replaced.
`AbaqusTarget.validate`	Instance method	Return validation issues for the Abaqus solver target.
`NativeTarget.to_dict`	Instance method	Serialize this specification to a versioned dictionary.
`NativeTarget.to_json`	Instance method	Serialize this specification to a versioned JSON document.
`NativeTarget.from_dict`	Class method	Create a validated solver target from a mapping payload.
`NativeTarget.from_json`	Class method	Create a validated specification from a JSON document.
`NativeTarget.copy_with`	Instance method	Return a validated copy with the supplied fields replaced.
`NativeTarget.validate`	Instance method	Return validation issues for the native solver target.
`CoreformIGAJobSpec.to_dict`	Instance method	Serialize this specification to a versioned dictionary.
`CoreformIGAJobSpec.to_json`	Instance method	Serialize this specification to a versioned JSON document.
`CoreformIGAJobSpec.from_dict`	Class method	Create a validated specification from a mapping payload.
`CoreformIGAJobSpec.from_json`	Class method	Create a validated specification from a JSON document.
`CoreformIGAJobSpec.copy_with`	Instance method	Return a validated copy with the supplied fields replaced.

Name	Kind	Description
`CoreformIGAJobSpec.validate`	Instance method	Return validation issues for the complete Job specification.
`MemoryUnits`	Class	Units used to interpret an Abaqus Job memory limit.
`ParallelMode`	Class	Process and thread modes supported by the Abaqus solver stage.
`Scheduler`	Class	Remote schedulers supported by portable Job submission settings.
`Local`	Class	Run a Job stage in the local process environment.
`RemoteQueue`	Class	Submit a Job stage through a remote scheduler queue.
`MeshStage`	Class	The Coreform IGA mesh build — the `coreform_iga_mesh` invocation. Alwayspresent and solver-independent.
`AbaqusInterop`	Class	The Coreform IGA -> Abaqus interop stage — the `coreform_interop_abaqus`invocation. Interop is solver-specific (it emits a solver's input deck), so itlives under the Abaqus solver target.
`AbaqusSolve`	Class	The Abaqus solve compute shape (Abaqus's mdb.Job / `abaqus job=` interface).These knobs are Abaqus-specific: another solver carries its own solve type withonly the resources that solver

Name	Kind	Description
		supports (see NativeSolve). `num_gpus -> numGPUs`and `parallel_mode -> multiprocessingMode` are per-field public-name overrides(Abaqus's own casing is irregular: `numCpus` but `numGPUs`).
`NativeSolve`	Class	The native Coreform IGA solve. Minimal by design: the native solver exposesfar fewer knobs than Abaqus, so it does NOT carry Abaqus's GPU / domain /parallel-mode / memory-limit fields. Fields are added here only as the nativesolver's run interface defines them.
`AbaqusTarget`	Class	Describe an Abaqus model, interop stage, and solve stage.
`NativeTarget`	Class	Describe a native Coreform IGA solver target.
`CoreformIGAJobSpec`	Class	Define a portable Coreform IGA Job pipeline.

7.1 `MemoryUnits`

Object path: `coreform.job.specs.MemoryUnits`

Units used to interpret an Abaqus Job memory limit.

7.1.a Attributes

Name	Type	Value	Description
`PERCENTAGE`	`MemoryUnits`	`'PERCENTAGE'`	Percentage of the physical mem-

Name	Type	Value	Description
			ory available to Abaqus.
`MEGA_BYTES`	`MemoryUnits`	`'MEGA_BYTES'`	Memory limit expressed in megabytes.
`GIGA_BYTES`	`MemoryUnits`	`'GIGA_BYTES'`	Memory limit expressed in gigabytes.

7.2 `ParallelMode`

Object path: `coreform.job.specs.ParallelMode`

Process and thread modes supported by the Abaqus solver stage.

7.2.a Attributes

Name	Type	Value	Description
`DEFAULT`	`ParallelMode`	`'DEFAULT'`	Let Abaqus select its default process and thread layout.
`MPI`	`ParallelMode`	`'MPI'`	Use one thread per MPI process.
`THREADS`	`ParallelMode`	`'THREADS'`	Use a single process with multiple threads.
`HYBRID`	`ParallelMode`	`'HYBRID'`	Combine MPI processes with multiple threads per process.

7.3 `Scheduler`

Object path: `coreform.job.specs.Scheduler`

Remote schedulers supported by portable Job submission settings.

7.3.a Attributes

Name	Type	Value	Description
`ABAQUS`	`Scheduler`	`'ABAQUS'`	Delegate queue submission to the Abaqus site environment.
`SLURM`	`Scheduler`	`'SLURM'`	Render SLURM scheduler directives.
`PBS`	`Scheduler`	`'PBS'`	Render PBS/OpenPBS scheduler directives.
`LSF`	`Scheduler`	`'LSF'`	Describe an LSF queue for downstream handling.

7.4 `Local`

Object path: `coreform.job.specs.Local`

Run a Job stage in the local process environment.

Call syntax

```
result = Local()
```

Signatures

```
Local()
```

Parameters

None.

7.4.a Attributes

Name	Type	Value	Description
`schemaVersion`	`int`	`0`	Version of the serialized specification schema.

Name	Type	Value	Description
`submission_type`	`str`	`'Local'`	Serialized discriminator for local execution.

7.4.b `validate`

Defined on: `coreform.job.specs.Local.validate`

Description: Return validation issues for this specification.

Call syntax

```
result = spec.validate()
```

Returns

Type	Description
`Tuple[ValidationIssue, ...]`	All detected issues, or an empty tuple when the specification is valid.

Signatures

```
validate() -> Tuple[ValidationIssue, ...]
```

Parameters

None.

7.4.c `to_dict`

Defined on: `coreform.job.specs.Local.to_dict`

Description: Serialize this specification to a versioned dictionary.

Call syntax

```
result = spec.to_dict()
```

Returns

Type	Description
`Dict[str, Any]`	Canonical specification fields with top-level schema metadata.

Signatures

```
to_dict() -> Dict[str, Any]
```

Parameters

None.

7.4.d `to_json`

Defined on: `coreform.job.specs.Local.to_json`

Description: Serialize this specification to a versioned JSON document.

Call syntax

```
result = spec.to_json()
```

Returns

Type	Description
`str`	Versioned JSON representation of this specification.

Signatures

```
to_json(indent: Optional[int] = 2) -> str
```

Parameters

Name	Type	Required	Default	Description
`indent`	`Optional[int]`	No	`2`	Number of spaces used to indent the JSON, or `None` for compact JSON.

7.4.e `from_dict`

Defined on: `coreform.job.specs.Local.from_dict`

Description: Create a validated submission specification from a mapping payload.

Call syntax

```
result = Local.from_dict(payload=payload)
```

Returns

Type	Description
`_SubmissionT`	Create a validated submission specification from a mapping payload.

Signatures

```
from_dict(payload: Mapping[str, Any]) -> Local
```

Parameters

Name	Type	Required	Default	Description
`payload`	`Mapping[str, Any]`	Yes	—	Submission fields and optional discriminator to validate and load.

7.4.f `from_json`

Defined on: `coreform.job.specs.Local.from_json`

Description: Create a validated specification from a JSON document.

Call syntax

```
result = Local.from_json(text=text)
```

Returns

Type	Description
`CoreformSpec`	Validated instance of the concrete specification class.

Signatures

```
from_json(text: str) -> Local
```

Parameters

Name	Type	Required	Default	Description
`text`	`str`	Yes	—	Versioned JSON specification document.

7.4.g `copy_with`

Defined on: `coreform.job.specs.Local.copy_with`

Description: Return a validated copy with the supplied fields replaced.

Call syntax

```
result = spec.copy_with(...)
```

Returns

Type	Description
`CoreformSpec`	Validated copy of the same concrete specification type.

Signatures

```
copy_with(**kwargs: Any) -> CoreformSpec
```

Parameters

Name	Type	Required	Default	Description
`kwargs`	`Any`	No	—	Specification fields and their replacement values.

7.5 `RemoteQueue`

Object path: `coreform.job.specs.RemoteQueue`

Submit a Job stage through a remote scheduler queue.

Call syntax

```
result = RemoteQueue()
```

Signatures

```
RemoteQueue(scheduler: Scheduler = Scheduler.ABAQUS, queue_name: str = '',
account: str = '', walltime: str = '', nodes: Optional[int] = None,
extra_directives: Tuple[str, ...] = ())
```

Parameters

Name	Type	Required	Default	Description
`scheduler`	`Scheduler`	No	`Scheduler.ABAQUS`	Scheduler responsible for the remote allocation.
`queue_name`	`str`	No	`''`	Site queue or partition name.
`account`	`str`	No	`''`	Scheduler account or project identifier.
`walltime`	`str`	No	`''`	Scheduler wall-time limit.
`nodes`	`Optional[int]`	No	`None`	Requested node count.
`extra_directives`	`Tuple[str, ...]`	No	`()`	Additional site-specific scheduler directive lines.

7.5.a Attributes

Name	Type	Value	Description
`schemaVersion`	`int`	`0`	Version of the serialized specification schema.
`scheduler`	`Scheduler`	`Scheduler.ABAQUS`	Scheduler responsible for the remote allocation.
`queue_name`	`str`	`''`	Site queue or partition name.
`account`	`str`	`''`	Scheduler account or project identifier.
`walltime`	`str`	`''`	Scheduler wall-time limit.
`nodes`	`Optional[int]`	`None`	Requested node count.

Name	Type	Value	Description
`extra_directives`	`Tuple[str, ...]`	`()`	Additional site-specific scheduler directive lines.
`submission_type`	`str`	`'RemoteQueue'`	Serialized discriminator for remote-queue execution.

7.5.b `to_dict`

Defined on: `coreform.job.specs.RemoteQueue.to_dict`

Description: Serialize this specification to a versioned dictionary.

Call syntax

```
result = spec.to_dict()
```

Returns

Type	Description
`Dict[str, Any]`	Canonical specification fields with top-level schema metadata.

Signatures

```
to_dict() -> Dict[str, Any]
```

Parameters

None.

7.5.c `to_json`

Defined on: `coreform.job.specs.RemoteQueue.to_json`

Description: Serialize this specification to a versioned JSON document.

Call syntax

```
result = spec.to_json()
```

Returns

Type	Description
`str`	Versioned JSON representation of this specification.

Signatures

```
to_json(indent: Optional[int] = 2) -> str
```

Parameters

Name	Type	Required	Default	Description
`indent`	`Optional[int]`	No	`2`	Number of spaces used to indent the JSON, or `None` for compact JSON.

7.5.d `from_dict`

Defined on: `coreform.job.specs.RemoteQueue.from_dict`

Description: Create a validated submission specification from a mapping payload.

Call syntax

```
result = RemoteQueue.from_dict(payload=payload)
```

Returns

Type	Description
`_SubmissionT`	Create a validated submission specification from a mapping payload.

Signatures

```
from_dict(payload: Mapping[str, Any]) -> RemoteQueue
```

Parameters

Name	Type	Required	Default	Description
`payload`	`Mapping[str, Any]`	Yes	—	Submission fields and op-

Name	Type	Required	Default	Description
				tional discriminator to validate and load.

7.5.e `from_json`

Defined on: `coreform.job.specs.RemoteQueue.from_json`

Description: Create a validated specification from a JSON document.

Call syntax

```
result = RemoteQueue.from_json(text=text)
```

Returns

Type	Description
`CoreformSpec`	Validated instance of the concrete specification class.

Signatures

```
from_json(text: str) -> RemoteQueue
```

Parameters

Name	Type	Required	Default	Description
`text`	`str`	Yes	—	Versioned JSON specification document.

7.5.f `copy_with`

Defined on: `coreform.job.specs.RemoteQueue.copy_with`

Description: Return a validated copy with the supplied fields replaced.

Call syntax

```
result = spec.copy_with(...)
```

Returns

Type	Description
`CoreformSpec`	Validated copy of the same concrete specification type.

Signatures

```
copy_with(**kwargs: Any) -> CoreformSpec
```

Parameters

Name	Type	Required	Default	Description
`kwargs`	`Any`	No	—	Specification fields and their replacement values.

7.5.g `validate`

Defined on: `coreform.job.specs.RemoteQueue.validate`

Description: Return validation issues for the remote queue settings.

Call syntax

```
result = spec.validate()
```

Returns

Type	Description
`Tuple[ValidationIssue, ...]`	Return validation issues for the remote queue settings.

Signatures

```
validate() -> Tuple[ValidationIssue, ...]
```

Parameters

None.

7.6 `MeshStage`

Object path: `coreform.job.specs.MeshStage`

The Coreform IGA mesh build — the `coreform_iga_mesh` invocation. Always present and solver-independent.

Call syntax

```
result = MeshStage()
```

Signatures

```
MeshStage(mesh_executable: str = '', mpiexec: str = '', num_cpus: int = 1, cf_file: Optional[str] = None, output_db: Optional[str] = None, parts: Tuple[str, ...] = (), extra_args: Tuple[str, ...] = (), queue: Optional[Union[Local, RemoteQueue]] = None)
```

Parameters

Name	Type	Required	Default	Description
`mesh_executable`	`str`	No	`''`	Coreform IGA Mesh executable name or path override.
`mpiexec`	`str`	No	`''`	MPI launcher name or path used when `num_cpus` is greater than one.
`num_cpus`	`int`	No	`1`	Positive CPU count for Mesh generation.
`cf_file`	`Optional[str]`	No	`None`	Coreform input filename, or `None` to derive it from the Job.
`output_db`	`Optional[str]`	No	`None`	Mesh database filename, or `None` to derive it from the input.

Name	Type	Required	Default	Description
`parts`	`Tuple[str, ...]`	No	`()`	Part names to mesh, or an empty tuple for all parts.
`extra_args`	`Tuple[str, ...]`	No	`()`	Additional command-line arguments passed to the Mesh executable.
`queue`	`Optional[Union[Local, RemoteQueue]]`	No	`None`	Submission policy for the Mesh stage.

7.6.a Attributes

Name	Type	Value	Description
`schemaVersion`	`int`	`0`	Version of the serialized specification schema.
`mesh_executable`	`str`	`''`	Coreform IGA Mesh executable name or path override.
`mpiexec`	`str`	`''`	MPI launcher name or path used when `num_cpus` is greater than one.
`num_cpus`	`int`	`1`	Positive CPU count for Mesh generation.
`cf_file`	`Optional[str]`	`None`	Coreform input filename, or `None` to derive it from the Job.

Name	Type	Value	Description
`output_db`	`Optional[str]`	`None`	Mesh database filename, or `None` to derive it from the input.
`parts`	`Tuple[str, ...]`	`()`	Part names to mesh, or an empty tuple for all parts.
`extra_args`	`Tuple[str, ...]`	`()`	Additional command-line arguments passed to the Mesh executable.
`queue`	`Optional[Union[Local, RemoteQueue]]`	`None`	Submission policy for the Mesh stage.

7.6.b `to_dict`

Defined on: `coreform.job.specs.MeshStage.to_dict`

Description: Serialize this specification to a versioned dictionary.

Call syntax

```
result = spec.to_dict()
```

Returns

Type	Description
`Dict[str, Any]`	Canonical specification fields with top-level schema metadata.

Signatures

```
to_dict() -> Dict[str, Any]
```

Parameters

None.

7.6.c `to_json`

Defined on: `coreform.job.specs.MeshStage.to_json`

Description: Serialize this specification to a versioned JSON document.

Call syntax

```
result = spec.to_json()
```

Returns

Type	Description
`str`	Versioned JSON representation of this specification.

Signatures

```
to_json(indent: Optional[int] = 2) -> str
```

Parameters

Name	Type	Required	Default	Description
`indent`	`Optional[int]`	No	`2`	Number of spaces used to indent the JSON, or `None` for compact JSON.

7.6.d `from_dict`

Defined on: `coreform.job.specs.MeshStage.from_dict`

Description: Create a validated specification from a mapping payload.

Call syntax

```
result = MeshStage.from_dict(payload=payload)
```

Returns

Type	Description
`CoreformSpec`	Validated instance of the concrete specification class.

Signatures

```
from_dict(payload: Mapping[str, Any]) -> MeshStage
```

Parameters

Name	Type	Required	Default	Description
`payload`	`Mapping[str, Any]`	Yes	—	Versioned specification fields to validate and load.

7.6.e `from_json`

Defined on: `coreform.job.specs.MeshStage.from_json`

Description: Create a validated specification from a JSON document.

Call syntax

```
result = MeshStage.from_json(text=text)
```

Returns

Type	Description
`CoreformSpec`	Validated instance of the concrete specification class.

Signatures

```
from_json(text: str) -> MeshStage
```

Parameters

Name	Type	Required	Default	Description
`text`	`str`	Yes	—	Versioned JSON specification document.

7.6.f `copy_with`

Defined on: `coreform.job.specs.MeshStage.copy_with`

Description: Return a validated copy with the supplied fields replaced.

Call syntax

```
result = spec.copy_with(...)
```

Returns

Type	Description
`CoreformSpec`	Validated copy of the same concrete specification type.

Signatures

```
copy_with(**kwargs: Any) -> CoreformSpec
```

Parameters

Name	Type	Required	Default	Description
`kwargs`	`Any`	No	—	Specification fields and their replacement values.

7.6.g `validate`

Defined on: `coreform.job.specs.MeshStage.validate`

Description: Return validation issues for the Mesh stage.

Call syntax

```
result = spec.validate()
```

Returns

Type	Description
`Tuple[ValidationIssue, ...]`	Return validation issues for the Mesh stage.

Signatures

```
validate() -> Tuple[ValidationIssue, ...]
```

Parameters

None.

7.7 `AbaqusInterop`

Object path: `coreform.job.specs.AbaqusInterop`

The Coreform IGA -> Abaqus interop stage — the `coreform_interop_abaqus` invocation. Interop is solver-specific (it emits a solver's input deck), so it lives under the Abaqus solver target.

Call syntax

```
result = AbaqusInterop()
```

Signatures

```
AbaqusInterop(interop_executable: str = '', cf_file: Optional[str] = None, sql_file: Optional[str] = None, inp_prefix: Optional[str] = None, parts: Tuple[str, ...] = (), dedup_epsilon: Optional[float] = None, keyword_report: Optional[str] = None, extra_args: Tuple[str, ...] = (), queue: Optional[Union[Local, RemoteQueue]] = None)
```

Parameters

Name	Type	Required	Default	Description
`interop_executable`	`str`	No	`''`	Coreform Abaqus interop executable name or path override.
`cf_file`	`Optional[str]`	No	`None`	Coreform input filename, or `None` to use the Mesh-stage input.
`sql_file`	`Optional[str]`	No	`None`	Mesh database filename, or `None` to use the Mesh-stage output.
`inp_prefix`	`Optional[str]`	No	`None`	Basename for the Abaqus input and result artifact family.

Name	Type	Required	Default	Description
`parts`	`Tuple[str, ...]`	No	`()`	Part names to translate, or an empty tuple for all parts.
`dedup_epsilon`	`Optional[float]`	No	`None`	Optional non-negative node-deduplication tolerance.
`keyword_report`	`Optional[str]`	No	`None`	Keyword-report filename, or `None` to derive it from the prefix.
`extra_args`	`Tuple[str, ...]`	No	`()`	Additional arguments passed to the interop executable.
`queue`	`Optional[Union[Local, RemoteQueue]]`	No	`None`	Submission policy for the interop stage.

7.7.a Attributes

Name	Type	Value	Description
`schemaVersion`	`int`	`0`	Version of the serialized specification schema.
`interop_executable`	`str`	`''`	Coreform Abaqus interop executable name or path override.
`cf_file`	`Optional[str]`	`None`	Coreform input filename, or `None` to use the Mesh-stage input.

Name	Type	Value	Description
`sql_file`	`Optional[str]`	`None`	Mesh database filename, or `None` to use the Mesh-stage output.
`inp_prefix`	`Optional[str]`	`None`	Basename for the Abaqus input and result artifact family.
`parts`	`Tuple[str, ...]`	`()`	Part names to translate, or an empty tuple for all parts.
`dedup_epsilon`	`Optional[float]`	`None`	Optional nonnegative node-deduplication tolerance.
`keyword_report`	`Optional[str]`	`None`	Keyword-report filename, or `None` to derive it from the prefix.
`extra_args`	`Tuple[str, ...]`	`()`	Additional arguments passed to the interop executable.
`queue`	`Optional[Union[Local, RemoteQueue]]`	`None`	Submission policy for the interop stage.

7.7.b `to_dict`

Defined on: `coreform.job.specs.AbaqusInterop.to_dict`

Description: Serialize this specification to a versioned dictionary.

Call syntax

```
result = spec.to_dict()
```

Returns

Type	Description
`Dict[str, Any]`	Canonical specification fields with top-level schema metadata.

Signatures

```
to_dict() -> Dict[str, Any]
```

Parameters

None.

7.7.c `to_json`

Defined on: `coreform.job.specs.AbaqusInterop.to_json`

Description: Serialize this specification to a versioned JSON document.

Call syntax

```
result = spec.to_json()
```

Returns

Type	Description
`str`	Versioned JSON representation of this specification.

Signatures

```
to_json(indent: Optional[int] = 2) -> str
```

Parameters

Name	Type	Required	Default	Description
`indent`	`Optional[int]`	No	`2`	Number of spaces used to indent the JSON, or `None` for compact JSON.

7.7.d `from_dict`

Defined on: `coreform.job.specs.AbaqusInterop.from_dict`

Description: Create a validated specification from a mapping payload.

Call syntax

```
result = AbaqusInterop.from_dict(payload=payload)
```

Returns

Type	Description
`CoreformSpec`	Validated instance of the concrete specification class.

Signatures

```
from_dict(payload: Mapping[str, Any]) -> AbaqusInterop
```

Parameters

Name	Type	Required	Default	Description
`payload`	`Mapping[str, Any]`	Yes	—	Versioned specification fields to validate and load.

7.7.e `from_json`

Defined on: `coreform.job.specs.AbaqusInterop.from_json`

Description: Create a validated specification from a JSON document.

Call syntax

```
result = AbaqusInterop.from_json(text=text)
```

Returns

Type	Description
`CoreformSpec`	Validated instance of the concrete specification class.

Signatures

```
from_json(text: str) -> AbaqusInterop
```

Parameters

Name	Type	Required	Default	Description
`text`	`str`	Yes	—	Versioned JSON specification document.

7.7.f `copy_with`

Defined on: `coreform.job.specs.AbaqusInterop.copy_with`

Description: Return a validated copy with the supplied fields replaced.

Call syntax

```
result = spec.copy_with(...)
```

Returns

Type	Description
`CoreformSpec`	Validated copy of the same concrete specification type.

Signatures

```
copy_with(**kwargs: Any) -> CoreformSpec
```

Parameters

Name	Type	Required	Default	Description
`kwargs`	`Any`	No	—	Specification fields and their replacement values.

7.7.g `validate`

Defined on: `coreform.job.specs.AbaqusInterop.validate`

Description: Return validation issues for the Abaqus interop stage.

Call syntax

```
result = spec.validate()
```

Returns

Type	Description
`Tuple[ValidationIssue, ...]`	Return validation issues for the Abaqus interop stage.

Signatures

```
validate() -> Tuple[ValidationIssue, ...]
```

Parameters

None.

7.8 `AbaqusSolve`

Object path: `coreform.job.specs.AbaqusSolve`

The Abaqus solve compute shape (Abaqus's mdb.Job / `abaqus job=` interface). These knobs are Abaqus-specific: another solver carries its own solve type with only the resources that solver supports (see NativeSolve). `num_gpus -> numGPUs` and `parallel_mode -> multiprocessingMode` are per-field public-name overrides (Abaqus's own casing is irregular: `numCpus` but `numGPUs`).

Call syntax

```
result = AbaqusSolve()
```

Signatures

```
AbaqusSolve(num_cpus: int = 1, num_gpus: int = 0, parallel_mode: ParallelMode = ParallelMode.THREADS, threads_per_process: Optional[int] = None, num_domains: Optional[int] = None, memory: int = 90, memory_units: MemoryUnits = MemoryUnits.PERCENTAGE, queue: Optional[Union[Local, RemoteQueue]] = None)
```

Parameters

Name	Type	Required	Default	Description
`num_cpus`	`int`	No	`1`	Positive CPU count for the Abaqus solve.
`num_gpus`	`int`	No	`0`	Nonnegative GPU count for the Abaqus solve.

Name	Type	Required	Default	Description
`parallel_mode`	`ParallelMode`	No	`ParallelMode.THREADS`	Solver process and thread mode.
`threads_per_process`	`Optional[int]`	No	`None`	Thread count per MPI process for an explicit hybrid layout.
`num_domains`	`Optional[int]`	No	`None`	Abaqus solver-domain count; it must be a multiple of `num_cpus`.
`memory`	`int`	No	`90`	Positive solver memory limit.
`memory_units`	`MemoryUnits`	No	`MemoryUnits.PERCENTAGE`	Units used to interpret `memory`.
`queue`	`Optional[Union[Local, RemoteQueue]]`	No	`None`	Submission policy for the solve stage.

7.8.a Attributes

Name	Type	Value	Description
`schemaVersion`	`int`	`0`	Version of the serialized specification schema.
`num_cpus`	`int`	`1`	Positive CPU count for the Abaqus solve.
`num_gpus`	`int`	`field(default=0, metadata={'cad_label': 'numGPUs'})`	Nonnegative GPU count for the Abaqus solve.

Name	Type	Value	Description
`parallel_mode`	`ParallelMode`	`field(default=(ParallelMode.THREADS), metadata={'camel_case': 'multiprocessingMode'})`	Solver process and thread mode.
`threads_per_process`	`Optional[int]`	`None`	Thread count per MPI process for an explicit hybrid layout.
`num_domains`	`Optional[int]`	`None`	Abaqus solver-domain count; it must be a multiple of `num_cpus`.
`memory`	`int`	`90`	Positive solver memory limit.
`memory_units`	`MemoryUnits`	`MemoryUnits.PERCENTAGE`	Units used to interpret `memory`.
`queue`	`Optional[Union[LocalQueue, RemoteQueue]]`	`None`	Submission policy for the solve stage.

7.8.b `to_dict`

Defined on: `coreform.job.specs.AbaqusSolve.to_dict`

Description: Serialize this specification to a versioned dictionary.

Call syntax

```
result = spec.to_dict()
```

Returns

Type	Description
`Dict[str, Any]`	Canonical specification fields with top-level schema metadata.

Signatures

```
to_dict() -> Dict[str, Any]
```

Parameters

None.

7.8.c `to_json`

Defined on: `coreform.job.specs.AbaqusSolve.to_json`

Description: Serialize this specification to a versioned JSON document.

Call syntax

```
result = spec.to_json()
```

Returns

Type	Description
`str`	Versioned JSON representation of this specification.

Signatures

```
to_json(indent: Optional[int] = 2) -> str
```

Parameters

Name	Type	Required	Default	Description
`indent`	`Optional[int]`	No	`2`	Number of spaces used to indent the JSON, or `None` for compact JSON.

7.8.d `from_dict`

Defined on: `coreform.job.specs.AbaqusSolve.from_dict`

Description: Create a validated specification from a mapping payload.

Call syntax

```
result = AbaqusSolve.from_dict(payload=payload)
```

Returns

Type	Description
`CoreformSpec`	Validated instance of the concrete specification class.

Signatures

```
from_dict(payload: Mapping[str, Any]) -> AbaqusSolve
```

Parameters

Name	Type	Required	Default	Description
`payload`	`Mapping[str, Any]`	Yes	—	Versioned specification fields to validate and load.

7.8.e `from_json`

Defined on: `coreform.job.specs.AbaqusSolve.from_json`

Description: Create a validated specification from a JSON document.

Call syntax

```
result = AbaqusSolve.from_json(text=text)
```

Returns

Type	Description
`CoreformSpec`	Validated instance of the concrete specification class.

Signatures

```
from_json(text: str) -> AbaqusSolve
```

Parameters

Name	Type	Required	Default	Description
`text`	`str`	Yes	—	Versioned JSON specification document.

7.8.f `copy_with`

Defined on: `coreform.job.specs.AbaqusSolve.copy_with`

Description: Return a validated copy with the supplied fields replaced.

Call syntax

```
result = spec.copy_with(...)
```

Returns

Type	Description
`CoreformSpec`	Validated copy of the same concrete specification type.

Signatures

```
copy_with(**kwargs: Any) -> CoreformSpec
```

Parameters

Name	Type	Required	Default	Description
`kwargs`	`Any`	No	—	Specification fields and their replacement values.

7.8.g `validate`

Defined on: `coreform.job.specs.AbaqusSolve.validate`

Description: Return validation issues for the Abaqus solve settings.

Call syntax

```
result = spec.validate()
```

Returns

Type	Description
`Tuple[ValidationIssue, ...]`	Return validation issues for the Abaqus solve settings.

Signatures

```
validate() -> Tuple[ValidationIssue, ...]
```

Parameters

None.

7.9 `NativeSolve`

Object path: `coreform.job.specs.NativeSolve`

The native Coreform IGA solve. Minimal by design: the native solver exposes far fewer knobs than Abaqus, so it does NOT carry Abaqus's GPU / domain / parallel-mode / memory-limit fields. Fields are added here only as the native solver's run interface defines them.

Call syntax

```
result = NativeSolve()
```

Signatures

```
NativeSolve(num_cpus: int = 1, queue: Optional[Union[Local, RemoteQueue]] = None)
```

Parameters

Name	Type	Required	Default	Description
`num_cpus`	`int`	No	`1`	Positive CPU count for the native solve.
`queue`	`Optional[Union[Local, RemoteQueue]]`	No	`None`	Submission policy for the native solve stage.

7.9.a Attributes

Name	Type	Value	Description
`schemaVersion`	`int`	`0`	Version of the serialized specification schema.
`num_cpus`	`int`	`1`	Positive CPU count for the native solve.
`queue`	`Optional[Union[Local, RemoteQueue]]`	`None`	Submission policy for the native solve stage.

7.9.b `to_dict`

Defined on: `coreform.job.specs.NativeSolve.to_dict`

Description: Serialize this specification to a versioned dictionary.

Call syntax

```
result = spec.to_dict()
```

Returns

Type	Description
`Dict[str, Any]`	Canonical specification fields with top-level schema metadata.

Signatures

```
to_dict() -> Dict[str, Any]
```

Parameters

None.

7.9.c `to_json`

Defined on: `coreform.job.specs.NativeSolve.to_json`

Description: Serialize this specification to a versioned JSON document.

Call syntax

```
result = spec.to_json()
```

Returns

Type	Description
`str`	Versioned JSON representation of this specification.

Signatures

```
to_json(indent: Optional[int] = 2) -> str
```

Parameters

Name	Type	Required	Default	Description
`indent`	`Optional[int]`	No	`2`	Number of spaces used to indent the

Name	Type	Required	Default	Description
				JSON, or `None` for compact JSON.

7.9.d `from_dict`

Defined on: `coreform.job.specs.NativeSolve.from_dict`

Description: Create a validated specification from a mapping payload.

Call syntax

```
result = NativeSolve.from_dict(payload=payload)
```

Returns

Type	Description
`CoreformSpec`	Validated instance of the concrete specification class.

Signatures

```
from_dict(payload: Mapping[str, Any]) -> NativeSolve
```

Parameters

Name	Type	Required	Default	Description
`payload`	`Mapping[str, Any]`	Yes	—	Versioned specification fields to validate and load.

7.9.e `from_json`

Defined on: `coreform.job.specs.NativeSolve.from_json`

Description: Create a validated specification from a JSON document.

Call syntax

```
result = NativeSolve.from_json(text=text)
```

Returns

Type	Description
`CoreformSpec`	Validated instance of the concrete specification class.

Signatures

```
from_json(text: str) -> NativeSolve
```

Parameters

Name	Type	Required	Default	Description
`text`	`str`	Yes	—	Versioned JSON specification document.

7.9.f `copy_with`

Defined on: `coreform.job.specs.NativeSolve.copy_with`

Description: Return a validated copy with the supplied fields replaced.

Call syntax

```
result = spec.copy_with(...)
```

Returns

Type	Description
`CoreformSpec`	Validated copy of the same concrete specification type.

Signatures

```
copy_with(**kwargs: Any) -> CoreformSpec
```

Parameters

Name	Type	Required	Default	Description
`kwargs`	`Any`	No	—	Specification fields and their replacement values.

7.9.g `validate`

Defined on: `coreform.job.specs.NativeSolve.validate`

Description: Return validation issues for the native solve settings.

Call syntax

```
result = spec.validate()
```

Returns

Type	Description
`Tuple[ValidationIssue, ...]`	Return validation issues for the native solve settings.

Signatures

```
validate() -> Tuple[ValidationIssue, ...]
```

Parameters

None.

7.10 `AbaqusTarget`

Object path: `coreform.job.specs.AbaqusTarget`

Describe an Abaqus model, interop stage, and solve stage.

Call syntax

```
result = AbaqusTarget(model=model, interop=interop, solve=solve)
```

Signatures

```
AbaqusTarget(model: str = None, interop: Optional[AbaqusInterop] = None, solve:
Optional[AbaqusSolve] = None)
```

Parameters

Name	Type	Required	Default	Description
`model`	`str`	Yes	—	Nonempty Abaqus model name.

Name	Type	Required	Default	Description
`interop`	`Optional[AbaqusInterop]`	Yes	—	Required Coreform-to-Abaqus interop stage.
`solve`	`Optional[AbaqusSolve]`	Yes	—	Required Abaqus solve stage.

7.10.a Attributes

Name	Type	Value	Description
`schemaVersion`	`int`	`0`	Version of the serialized specification schema.
`model`	`str`	`None`	Nonempty Abaqus model name.
`interop`	`Optional[AbaqusInterop]`	`field(default=None, metadata={'required': True})`	Required Coreform-to-Abaqus interop stage.
`solve`	`Optional[AbaqusSolve]`	`field(default=None, metadata={'required': True})`	Required Abaqus solve stage.
`solver_type`	`str`	`'AbaqusTarget'`	Serialized discriminator for an Abaqus target.

7.10.b `to_dict`

Defined on: `coreform.job.specs.AbaqusTarget.to_dict`

Description: Serialize this specification to a versioned dictionary.

Call syntax

```
result = spec.to_dict()
```

Returns

Type	Description
`Dict[str, Any]`	Canonical specification fields with top-level schema metadata.

Signatures

```
to_dict() -> Dict[str, Any]
```

Parameters

None.

7.10.c `to_json`

Defined on: `coreform.job.specs.AbaqusTarget.to_json`

Description: Serialize this specification to a versioned JSON document.

Call syntax

```
result = spec.to_json()
```

Returns

Type	Description
`str`	Versioned JSON representation of this specification.

Signatures

```
to_json(indent: Optional[int] = 2) -> str
```

Parameters

Name	Type	Required	Default	Description
`indent`	`Optional[int]`	No	`2`	Number of spaces used to indent the JSON, or `None` for compact JSON.

7.10.d `from_dict`

Defined on: `coreform.job.specs.AbaqusTarget.from_dict`

Description: Create a validated solver target from a mapping payload.

Call syntax

```
result = AbaqusTarget.from_dict(payload=payload)
```

Returns

Type	Description
`_SolverTargetT`	Create a validated solver target from a mapping payload.

Signatures

```
from_dict(payload: Mapping[str, Any]) -> AbaqusTarget
```

Parameters

Name	Type	Required	Default	Description
`payload`	`Mapping[str, Any]`	Yes	—	Solver-target fields and optional discriminator to validate and load.

7.10.e `from_json`

Defined on: `coreform.job.specs.AbaqusTarget.from_json`

Description: Create a validated specification from a JSON document.

Call syntax

```
result = AbaqusTarget.from_json(text=text)
```

Returns

Type	Description
`CoreformSpec`	Validated instance of the concrete specification class.

Signatures

```
from_json(text: str) -> AbaqusTarget
```

Parameters

Name	Type	Required	Default	Description
`text`	`str`	Yes	—	Versioned JSON specification document.

7.10.f `copy_with`

Defined on: `coreform.job.specs.AbaqusTarget.copy_with`

Description: Return a validated copy with the supplied fields replaced.

Call syntax

```
result = spec.copy_with(...)
```

Returns

Type	Description
`CoreformSpec`	Validated copy of the same concrete specification type.

Signatures

```
copy_with(**kwargs: Any) -> CoreformSpec
```

Parameters

Name	Type	Required	Default	Description
`kwargs`	`Any`	No	—	Specification fields and their replacement values.

7.10.g `validate`

Defined on: `coreform.job.specs.AbaqusTarget.validate`

Description: Return validation issues for the Abaqus solver target.

Call syntax

```
result = spec.validate()
```

Returns

Type	Description
`Tuple[ValidationIssue, ...]`	Return validation issues for the Abaqus solver target.

Signatures

```
validate() -> Tuple[ValidationIssue, ...]
```

Parameters

None.

7.11 `NativeTarget`

Object path: `coreform.job.specs.NativeTarget`

Describe a native Coreform IGA solver target.

Call syntax

```
result = NativeTarget(solve=solve)
```

Signatures

```
NativeTarget(solve: Optional[NativeSolve] = None)
```

Parameters

Name	Type	Required	Default	Description
`solve`	`Optional[NativeSolve]`	Yes	—	Required native Coreform IGA solve stage.

7.11.a Attributes

Name	Type	Value	Description
`schemaVersion`	`int`	`0`	Version of the serialized specification schema.

Name	Type	Value	Description
`solve`	`Optional[NativeSolve]`	`field(default=None, metadata={'required': True})`	Required native Coreform IGA solve stage.
`solver_type`	`str`	`'NativeTarget'`	Serialized discriminator for a native target.

7.11.b `to_dict`

Defined on: `coreform.job.specs.NativeTarget.to_dict`

Description: Serialize this specification to a versioned dictionary.

Call syntax

```
result = spec.to_dict()
```

Returns

Type	Description
`Dict[str, Any]`	Canonical specification fields with top-level schema metadata.

Signatures

```
to_dict() -> Dict[str, Any]
```

Parameters

None.

7.11.c `to_json`

Defined on: `coreform.job.specs.NativeTarget.to_json`

Description: Serialize this specification to a versioned JSON document.

Call syntax

```
result = spec.to_json()
```

Returns

Type	Description
`str`	Versioned JSON representation of this specification.

Signatures

```
to_json(indent: Optional[int] = 2) -> str
```

Parameters

Name	Type	Required	Default	Description
`indent`	`Optional[int]`	No	`2`	Number of spaces used to indent the JSON, or `None` for compact JSON.

7.11.d `from_dict`

Defined on: `coreform.job.specs.NativeTarget.from_dict`

Description: Create a validated solver target from a mapping payload.

Call syntax

```
result = NativeTarget.from_dict(payload=payload)
```

Returns

Type	Description
`_SolverTargetT`	Create a validated solver target from a mapping payload.

Signatures

```
from_dict(payload: Mapping[str, Any]) -> NativeTarget
```

Parameters

Name	Type	Required	Default	Description
`payload`	`Mapping[str, Any]`	Yes	—	Solver-target fields and op-

Name	Type	Required	Default	Description
				tional discriminator to validate and load.

7.11.e `from_json`

Defined on: `coreform.job.specs.NativeTarget.from_json`

Description: Create a validated specification from a JSON document.

Call syntax

```
result = NativeTarget.from_json(text=text)
```

Returns

Type	Description
`CoreformSpec`	Validated instance of the concrete specification class.

Signatures

```
from_json(text: str) -> NativeTarget
```

Parameters

Name	Type	Required	Default	Description
`text`	`str`	Yes	—	Versioned JSON specification document.

7.11.f `copy_with`

Defined on: `coreform.job.specs.NativeTarget.copy_with`

Description: Return a validated copy with the supplied fields replaced.

Call syntax

```
result = spec.copy_with(...)
```

Returns

Type	Description
`CoreformSpec`	Validated copy of the same concrete specification type.

Signatures

```
copy_with(**kwargs: Any) -> CoreformSpec
```

Parameters

Name	Type	Required	Default	Description
`kwargs`	`Any`	No	—	Specification fields and their replacement values.

7.11.g `validate`

Defined on: `coreform.job.specs.NativeTarget.validate`

Description: Return validation issues for the native solver target.

Call syntax

```
result = spec.validate()
```

Returns

Type	Description
`Tuple[ValidationIssue, ...]`	Return validation issues for the native solver target.

Signatures

```
validate() -> Tuple[ValidationIssue, ...]
```

Parameters

None.

7.12 `CoreformIGAJobSpec`

Object path: `coreform.job.specs.CoreformIGAJobSpec`

Define a portable Coreform IGA Job pipeline.

Call syntax

```
result = CoreformIGAJobSpec(name=name, mesh=mesh, solver=solver)
```

Signatures

```
CoreformIGAJobSpec(name: str = None, description: str = '', mesh:
Optional[MeshStage] = None, solver: Optional[Union[AbaqusTarget, NativeTarget]]
= None)
```

Parameters

Name	Type	Required	Default	Description
`name`	`str`	Yes	—	Unique, nonempty Job name.
`description`	`str`	No	`''`	Optional human-readable Job description.
`mesh`	`Optional[MeshStage]`	Yes	—	Required Coreform IGA Mesh stage.
`solver`	`Optional[Union[AbaqusTarget, NativeTarget]]`	Yes	—	Required downstream solver target.

7.12.a Attributes

Name	Type	Value	Description
`schemaVersion`	`int`	`0`	Version of the serialized specification schema.
`name`	`str`	`None`	Unique, nonempty Job name.
`description`	`str`	`''`	Optional human-readable Job description.

Name	Type	Value	Description
`mesh`	`Optional[MeshStage]`	`field(default=None, metadata={'required': True})`	Required Coreform IGA Mesh stage.
`solver`	`Optional[Union[AbaqusTarget, NativeTarget]]`	`field(default=None, metadata={'required': True})`	Required downstream solver target.

7.12.b `to_dict`

Defined on: `coreform.job.specs.CoreformIGAJobSpec.to_dict`

Description: Serialize this specification to a versioned dictionary.

Call syntax

```
result = spec.to_dict()
```

Returns

Type	Description
`Dict[str, Any]`	Canonical specification fields with top-level schema metadata.

Signatures

```
to_dict() -> Dict[str, Any]
```

Parameters

None.

7.12.c `to_json`

Defined on: `coreform.job.specs.CoreformIGAJobSpec.to_json`

Description: Serialize this specification to a versioned JSON document.

Call syntax

```
result = spec.to_json()
```

Returns

Type	Description
`str`	Versioned JSON representation of this specification.

Signatures

```
to_json(indent: Optional[int] = 2) -> str
```

Parameters

Name	Type	Required	Default	Description
`indent`	`Optional[int]`	No	`2`	Number of spaces used to indent the JSON, or `None` for compact JSON.

7.12.d `from_dict`

Defined on: `coreform.job.specs.CoreformIGAJobSpec.from_dict`

Description: Create a validated specification from a mapping payload.

Call syntax

```
result = CoreformIGAJobSpec.from_dict(payload=payload)
```

Returns

Type	Description
`CoreformSpec`	Validated instance of the concrete specification class.

Signatures

```
from_dict(payload: Mapping[str, Any]) -> CoreformIGAJobSpec
```

Parameters

Name	Type	Required	Default	Description
`payload`	`Mapping[str, Any]`	Yes	—	Versioned specification

Name	Type	Required	Default	Description
				fields to validate and load.

7.12.e `from_json`

Defined on: `coreform.job.specs.CoreformIGAJobSpec.from_json`

Description: Create a validated specification from a JSON document.

Call syntax

```
result = CoreformIGAJobSpec.from_json(text=text)
```

Returns

Type	Description
`CoreformSpec`	Validated instance of the concrete specification class.

Signatures

```
from_json(text: str) -> CoreformIGAJobSpec
```

Parameters

Name	Type	Required	Default	Description
`text`	`str`	Yes	—	Versioned JSON specification document.

7.12.f `copy_with`

Defined on: `coreform.job.specs.CoreformIGAJobSpec.copy_with`

Description: Return a validated copy with the supplied fields replaced.

Call syntax

```
result = spec.copy_with(...)
```

Returns

Type	Description
`CoreformSpec`	Validated copy of the same concrete specification type.

Signatures

```
copy_with(**kwargs: Any) -> CoreformSpec
```

Parameters

Name	Type	Required	Default	Description
`kwargs`	`Any`	No	—	Specification fields and their replacement values.

7.12.g `validate`

Defined on: `coreform.job.specs.CoreformIGAJobSpec.validate`

Description: Return validation issues for the complete Job specification.

Call syntax

```
result = spec.validate()
```

Returns

Type	Description
`Tuple[ValidationIssue, ...]`	Return validation issues for the complete Job specification.

Signatures

```
validate() -> Tuple[ValidationIssue, ...]
```

Parameters

None.