Getting started with scripting

Table of contents

Coreform IGA scripts run inside the Abaqus Python kernel supplied by the Coreform IGA custom application. The `coreform_iga` launcher loads Abaqus/CAE and the Coreform kernel data classes before executing your script.

1 Run a script from a terminal

Save your script before launching it and use an absolute path when practical. The following examples use the default Coreform IGA for Abaqus 2026.9 installation locations.

On Linux:

```
cd /home/jsmith/cifa-scripts

/opt/Coreform-IGA-2026.9/bin/coreform_iga \
  noGUI="/home/jsmith/cifa-scripts/my_script.py"
```

If the launcher is on `PATH`, you can use:

```
coreform_iga noGUI="$PWD/my_script.py"
```

On Windows PowerShell:

```
cd 'C:\Users\jsmith\cifa-scripts'

& 'C:\Program Files\Coreform IGA 2026.9\coreform_iga.bat' `
  'noGUI=C:\Users\jsmith\cifa-scripts\my_script.py'
```

Replace the example username, version, and installation directory with the paths on your computer. On Linux, `command -v coreform_iga` reports the launcher selected from `PATH`. On Windows PowerShell, use `Get-Command coreform_iga.bat`.

The script runs inside Abaqus Python—not the system Python interpreter. Its `print()` output and Abaqus or Coreform diagnostics appear in the terminal.

> ! Important
>
> Use the Coreform `coreform_iga` launcher. A plain Abaqus/CAE session does not load the Coreform IGA kernel data classes.

2 Initialize the model database

Start scripts by importing the current Abaqus model database and initializing its Coreform namespaces:

```
from abaqus import mdb
import coreform.abaqus

coreform.abaqus.init_iga_mdb()

model = mdb.models["Model-1"]
probeApi = model.steps.customData.CoreformIGA
meshApi = model.parts["Part-1"].customData.CoreformIGA
jobApi = mdb.customData.CoreformIGA
```

`init_iga_mdb()` is idempotent. Call it again after creating models, parts, or assembly instances to initialize the new objects. For one part or independent instance created later, `init_iga_mesh_owner(owner)` attaches only that owner's Mesh namespace.

The persistent object model has three domain entry points:

Domain	Namespace	Stored records
Probes	`model.steps.customData.CoreformIGA`	`namespace.probes[name]`
Meshes	`part.customData.CoreformIGA` or `independentInstance.customData.CoreformIGA`	`namespace.mesh`
Jobs	`mdb.customData.CoreformIGA`	`namespace.jobs[name]`

3 Understand the API layers

The `coreform` package separates portable Coreform data from integrations with specific downstream applications:

- `coreform.job`, `coreform.mesh`, and `coreform.probe` form the portable Coreform layer. These modules define the raw, immutable specifications and related operations using conventional Python snake_case names. They do not depend on Abaqus and can be used by ordinary Python programs.
- `coreform.abaqus.job`, `coreform.abaqus.mesh`, and `coreform.abaqus.probe` form the Abaqus adapter layer. These modules translate portable data into Abaqus-facing records and operations, including persistent `customData` objects, owner resolution, validation, and journaling. Their customer-facing operations follow Abaqus naming conventions.

The Scripting Reference mirrors the `interop/abaqus/cf_app/coreform` source tree. Each page represents one source directory: its linked directory tree leads to adapter or portable subpackages, and its file sections document public Python declarations defined directly in that directory.

4 Create and change persistent data

Abaqus-facing factories use PascalCase names such as `PointProbe` and `ImmersedRectilinearMesh`. Their parameters and persistent-record methods use Abaqus-style camelCase names such as `instanceNames`, `elementSize`, and `setValues`.

Factories return the newly created persistent record. Record attributes are read-only; use `setValues()` so validation, persistence, and Abaqus journaling remain synchronized. Use `deleteProbe()`, `deleteMesh()`, and `deleteJob()` rather than deleting repository entries directly.

Invalid values raise `coreform.validation.ValidationError`. A failed creation or update is validated before the stored data changes.

5 Work with portable specifications

Portable specifications are immutable, Abaqus-free Python objects that use snake_case names. They can be authored outside Abaqus and passed to an Abaqus factory through its mutually exclusive `spec=` form:

```
from coreform.probe import PointProbeSpec, ProbeRegion

spec = PointProbeSpec(
    name="Probe-1",
    region=ProbeRegion.instances(("PART-1-1",)),
    location=(0.0, 0.0, 0.0),
)

probe = probeApi.PointProbe(spec=spec)
```

Every persistent Probe, Mesh, and Job record provides `toSpec()` to recover its portable representation. Dictionaries and JSON are the supported versioned interchange formats for moving specifications between processes or downstream tools.

5.1 Author portable specifications outside Abaqus

The portable modules are shipped as Python source inside the installed `cf_app` directory; they are not currently distributed as a separate `pip` package. Add that directory to `PYTHONPATH` when running an ordinary Python 3.10 interpreter. Import only `coreform.probe`, `coreform.mesh`, `coreform.job`, `coreform.types`, and `coreform.validation` in this environment; `coreform.abaqus` requires the Abaqus runtime.

For the default Linux installation:

```
export COREFORM_IGA_CF_APP=/opt/Coreform-IGA-2026.9/interop/abaqus/cf_app
export PYTHONPATH="$COREFORM_IGA_CF_APP${PYTHONPATH:+:$PYTHONPATH}"

python3.10 make_specs.py
```

For the default Windows installation in PowerShell:

```
$coreformIgaCfApp = 'C:\Program Files\Coreform IGA 2026.9\interop\abaqus\cf_app'
$env:PYTHONPATH = "$coreformIgaCfApp;$env:PYTHONPATH"

py -3.10 .\make_specs.py
```

Replace the example version and installation directory when needed. From a source checkout, use the checkout's `interop/abaqus/cf_app` directory instead.

An ordinary Python script can create and serialize a specification without importing Abaqus:

```
from pathlib import Path

from coreform.mesh import ElementSizeSizing, ImmersedRectilinearMeshSpec

spec = ImmersedRectilinearMeshSpec(
    degree=2,
    continuity=1,
    sizing=ElementSizeSizing((0.5, 0.5, 0.5)),
)
Path("mesh-spec.json").write_text(spec.to_json(), encoding="utf-8")
```

Load the versioned JSON inside a script launched with `coreform_iga`, then pass the reconstructed object through the explicit `spec=` form:

```
from pathlib import Path

from coreform.mesh import ImmersedRectilinearMeshSpec

spec = ImmersedRectilinearMeshSpec.from_json(
    Path("mesh-spec.json").read_text(encoding="utf-8")
)
mesh = meshApi.ImmersedRectilinearMesh(spec=spec)
```

Pickle can also reconstruct these objects across trusted, compatible Python environments because both use the same `coreform.*` module paths. Do not unpickle untrusted input. Pickle is not a stable contract across Python protocols or Coreform releases and cannot serve non-Python tools; use the versioned dictionary or JSON representation for durable interchange.

6 Verify persistent data

After building or reopening a model database, you can validate all Coreform namespaces and records without modifying them:

```
result = coreform.abaqus.verify_iga_mdb()

if not result.passed:
    for issue in result.issues:
        print(issue)
```

Continue with Working with Probes, Working with Meshes, or Running Jobs. For exact method details, use the Scripting Reference.