Important

IBM Quantum Platform is moving and this version will be sunset on July 1. To get started on the new platform, read the migration guide.

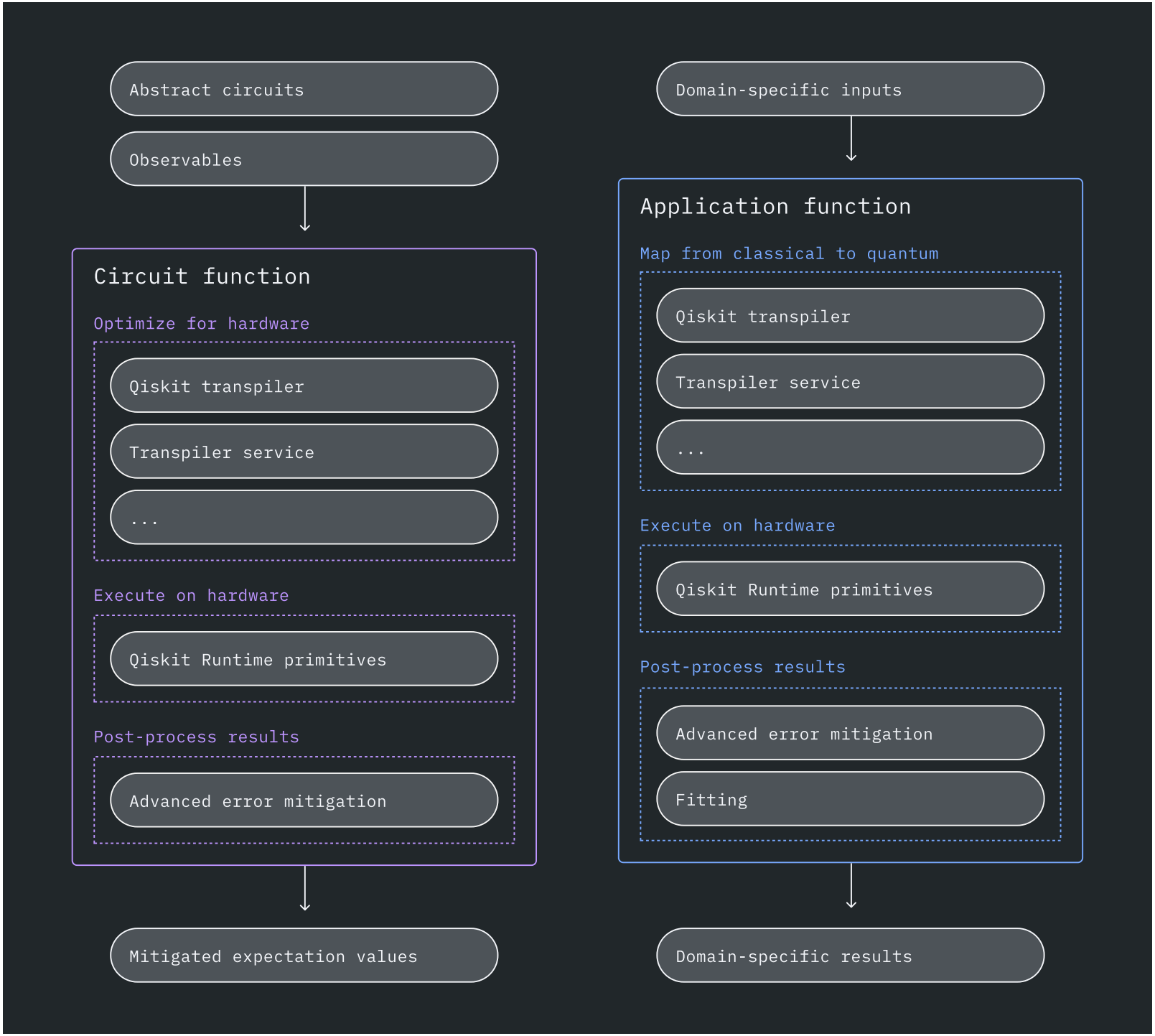
Introduction to Qiskit Functions

Notes

- This documentation is relevant to IBM Quantum® Platform Classic. If you need the newer version, go to the new IBM Quantum Platform documentation. ↗
- Qiskit Functions are an experimental feature available only to IBM Quantum® Premium Plan users. They are in preview release status and subject to change.

Package versions

Qiskit Functions simplify and accelerate utility-scale algorithm discovery and application development, by abstracting away parts of the quantum software development workflow. In this way, Qiskit Functions free up time normally spent hand-writing code and fine-tuning experiments.


Abstract circuits
Observables
Circuit function
Optimize for hardware
Qiskit transpiler
Transpiler service
...
Execute on hardware
Qiskit Runtime primitives
Post-process results
Advanced error mitigation
Mitigated expectation values
Domain-specific inputs
Application function
Map from classical to quantum
Qiskit transpiler
Transpiler service
...
Execute on hardware
Qiskit Runtime primitives
Post-process results
Advanced error mitigation
Fitting
Domain-specific results

Functions come in two forms:

Type	What does it do?	Example inputs and outputs	Who is it for?
Circuit function	Simplified interface for running circuits. Abstracts transpilation, error suppression, error mitigation	**Input**: Abstract PUBs **Output**: Mitigated expectation values	Researchers using Qiskit to discover new algorithms and applications, without needing to focus on optimizing for hardware or handling error. Circuit functions can be used to build custom application functions.
Application function	Covers higher-level tasks, like exploring algorithms and domain-specific use cases. Abstracts quantum workflow to solve tasks, with classical inputs and outputs	**Input**: Molecules, graphs **Output**: Energy, cost	Researchers in non-quantum domains, integrating quantum into existing large-scale classical workflows, without needing to map classical data to quantum circuits.

Functions are provided by IBM® and third-party partners. Each is performant for specific workload characteristics and have unique performance-tuning options. Premium Plan users can get started with IBM Qiskit Functions for free, or procure a license from one of the partners who have contributed a function to the catalog.

Get started with Qiskit Functions

Install Qiskit Functions Catalog client

1. To start using Qiskit Functions, install the IBM Qiskit Functions Catalog client:

```
pip install qiskit-ibm-catalog
```

2. Retrieve your API key from the IBM Quantum account page ↗, and activate your Python virtual environment. See the installation instructions if you do not already have a virtual environment set up.

If you are working in a trusted Python environment (such as on a personal laptop or workstation), use the `save_account()` method to save your credentials locally. (Skip to the next step if you are not using a trusted environment, such as a shared or public computer, to authenticate to IBM Quantum Platform.)

To use `save_account()`, run `python` in your shell, then enter the following:

```
from qiskit_ibm_catalog import QiskitFunctionsCatalog

QiskitFunctionsCatalog.save_account(token="<your-token>")
```

Type `exit()`. From now on, whenever you need to authenticate to the service, you can load your credentials with `QiskitFunctionsCatalog()`.

```
# Load saved credentials
from qiskit_ibm_catalog import QiskitFunctionsCatalog

catalog = QiskitFunctionsCatalog()
```

3. **Avoid executing code on an untrusted machine or an external cloud Python environment to minimize security risks.** If you must use an untrusted environment (on, for example, a public computer), change your API key after each use by expiring it on the IBM Quantum Platform dashboard ↗ (click the refresh button in the API key field) to reduce risk. To initialize the service in this situation, expand the following section to view code you can use:

   Initialize the service in an untrusted environment

> **Caution**
>
> **Protect your API key!** Never include your key in source code, Python script, or notebook file. When sharing code with others, ensure that your API key is not embedded directly within the Python script. Instead, share the script without the key and provide instructions for securely setting it up.
>
> If you accidentally share your key with someone or include it in version control like Git, immediately revoke your key by expiring it on the IBM Quantum Platform dashboard ↗ (click the refresh button in the API key field) to reduce risk.

4. After you have authenticated, you can list the functions from the Qiskit Functions Catalog that you have access to:

```
catalog.list()
```

Output:

```
[QiskitFunction(qunova/hivqe-chemistry),
 QiskitFunction(qunasys/quri-chemistry),
 QiskitFunction(algorithmiq/tem),
 QiskitFunction(algorithmiq/tem-gpu),
 QiskitFunction(qedma/qesem),
 QiskitFunction(multiverse/singularity),
 QiskitFunction(ibm/circuit-function),
 QiskitFunction(q-ctrl/optimization-solver),
 QiskitFunction(q-ctrl/performance-management),
 QiskitFunction(kipu-quantum/iskay-quantum-optimizer),
```

Run enabled functions

After a catalog object has been instantiated, you can select a function using `catalog.load(provider/function-name)`:

```
ibm_cf = catalog.load("ibm/circuit-function")
```

Each Qiskit Function has custom inputs, options, and outputs. Check the specific documentation pages for the function you want to run for more information. By default, all users can only run one function job at a time:

```
job = ibm_cf.run(
    pubs=[(circuit, observable)],
    instance=instance,
    backend_name=backend_name,  # E.g. "ibm_fez"
)

job.job_id
```

Output:

```
'6054eb66-f2f2-48d4-9f68-333bd8559379'
```

Check job status

> **Tip**
>
> Currently, the IBM Quantum workloads table only reflects Qiskit Runtime workloads. Use `job.status()` to see your Qiskit Function workload's current status.

With your Qiskit Function `job_id`, you can check the status of running jobs. This includes the following statuses:

- `QUEUED`: The remote program is in the Qiskit Function queue. The queue priority is based on how much you've used Qiskit Functions.
- `INITIALIZING`: The remote program is starting; this includes setting up the remote environment and installing dependencies.
- `RUNNING`: The program is running.
- `DONE`: The program is complete, and you can retrieve result data with `job.results()`.
- `ERROR`: The program stopped running because of a problem. Use `job.result()` to get the error message.
- `CANCELED`: The program was canceled; either by a user, the service, or the server.

```
job.status()
```

Output:

```
'QUEUED'
```

Retrieve results

After a program is `DONE`, you can use `job.results()` to fetch the result. This output format varies with each function, so be sure to follow the specific documentation:

```
result = job.result()
print(result)
```

Output:

```
PrimitiveResult([PubResult(data=DataBin(evs=np.ndarray(<shape=(), dtype=float64>), stc
```

You can also cancel a job at any time:

```
job.stop()
```

Output:

```
'Job has been stopped.'
```

List previously run jobs run with Qiskit Functions

You can use `jobs()` to list all jobs submitted to Qiskit Functions:

```
old_jobs = catalog.jobs()
old_jobs
```

Output:

```
[<Job | 6054eb66-f2f2-48d4-9f68-333bd8559379>,
 <Job | a7464168-eff6-4676-a481-ea4d2dd1a13b>,
 <Job | 2765f0c1-7973-48f6-a5c2-746141f51761>,
 <Job | 75b30e22-0953-4c83-9ff7-a5797ffaddcf>,
 <Job | 93018ba6-40b2-439e-a517-1c05e26bef01>,
 <Job | fe82a257-ea09-4229-9189-feea41fc5481>,
 <Job | 466e9122-16b5-493b-9006-7456970663cd>,
 <Job | 5fd4eb03-0fb8-4ace-a4f2-33cb18025f9b>,
 <Job | 7b958091-600e-449a-a167-75fe11075c9d>,
 <Job | eaa2db93-0d2f-4b52-9f1b-a69fc03dc3d6>]
```

If you already have the job ID for a certain job, you can retrieve the job with `catalog.get_job_by_id()`:

```
# First, get the most recent job that has been executed.
latest_job = old_jobs[0]

# We can also get that same job with get_job_by_id
job_by_id = catalog.get_job_by_id(latest_job.job_id)

# Verify that the job is the same using both retrieval methods.
assert job_by_id == latest_job

# Print the job_id for this job.
print(job_by_id.job_id)
```

Fetch error messages

If a program status is `ERROR`, use `job.result()` to fetch the error message as follows:

```
print(job.result())
```

Output:

```
PrimitiveResult([PubResult(data=DataBin(evs=np.ndarray(<shape=(), dtype=float64>), stc
```

Next steps

Recommendations

- Explore circuit functions to build new algorithms and applications, without needing to manage transpilation or error handling.
- Explore application functions to solve domain-specific tasks, with classical inputs and outputs.

Was this page helpful?

Yes No

Report a bug or request content on GitHub ↗.