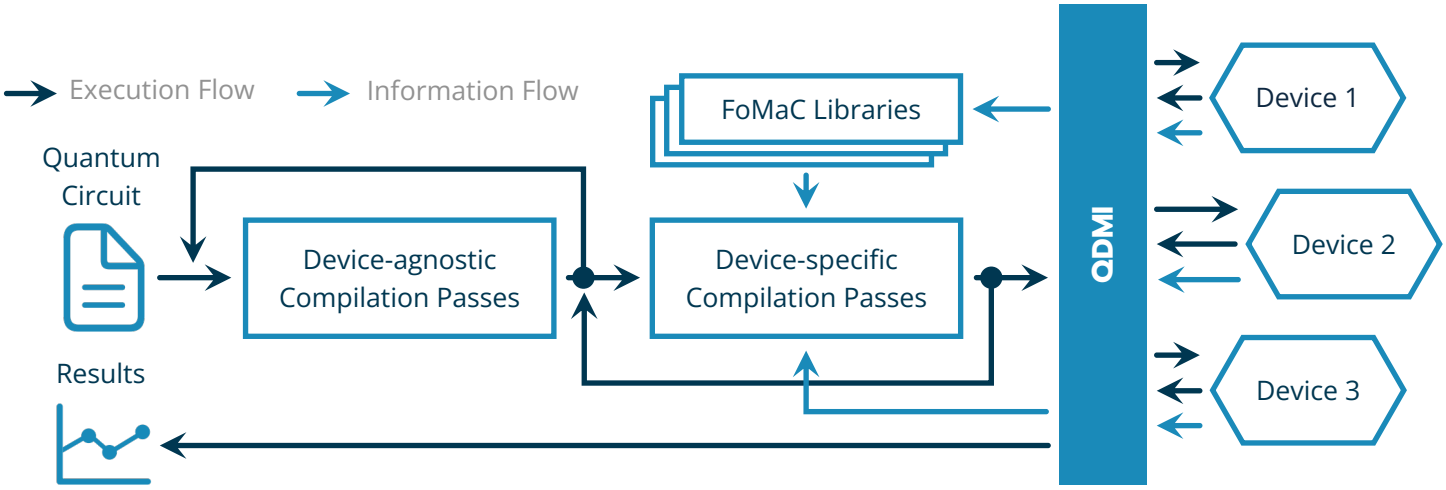
QDMI – Quantum Device Management Interface

Hardware-Software Interface for the Munich Quantum Software Stack (MQSS)

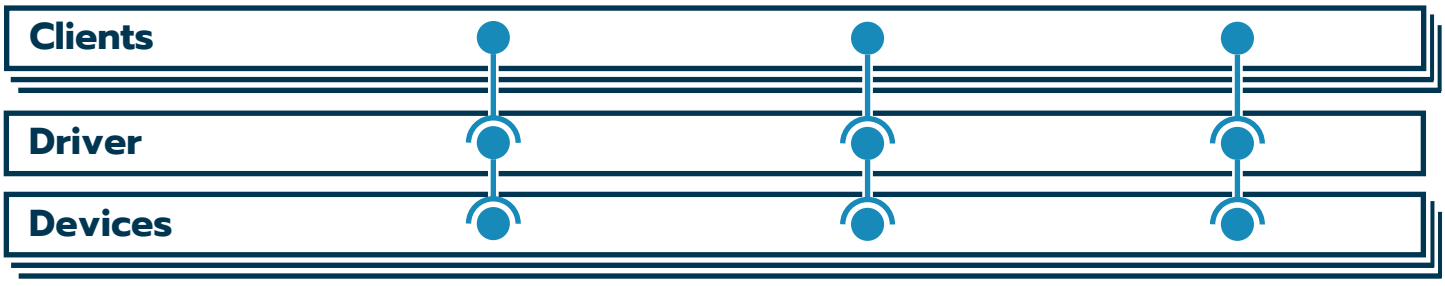
munich-quantum-software-stack.github.io/QDMI



The Quantum Device Management Interface (QDMI) is the central part of the Munich Quantum Software Stack (MQSS)—a sophisticated software stack to connect end users to a wide range of possible quantum backends. It enables the submission to and the control of gate-based quantum systems and enables software tools to automatically retrieve and adapt to changing physical characteristics and constraints of different platforms. QDMI strives to connect the software and hardware developers, mediating between their competing interests, bridging between technologies, and eventually providing corresponding figures of merits and constraints to be considered. QDMI is, therefore, the method of choice for integrating new platforms into the MQSS and for software tools to query information from these platforms. QDMI is provided as a header-only library in C to allow fast integration into an HPC environment and consists of three different entities: clients, devices, and a driver. These entities interact via two interfaces: client and device interface, each of which is subdivided into three parts: session, query, and job interface.

A *consumes*,
B *implements* the interface

Session	Query	Control
• User Management	• Device Properties	• Job Configuration
• Access Control	• Site Properties	• Job Submission
• Resource Management	• Operation Properties	• Result Retrieval



Technical Contact
Lukas Burgholzer: lukas.burgholzer@tum.de
Yannick Stade: yannick.stade@tum.de

General Contact
Robert Wille, Martin Schulz, Jorge Echavarria
mqss@munich-quantum-valley.de

Example Device

The following code shows a snippet from an example device implementation. Here, the function to query device properties is shown. Devices prefix every QDMI function with their own prefix, in this example *EX*. The types of sites (also operations and jobs) are implemented by the device. The code assumes, that all device's sites are contained in some array named *SITES*.

```
int EX_ QDMI_device_session_query_device_property(EX_QDMI_Device_Session session,
      const QDMI_Device_Property prop, const size_t size, void *value, size_t *size_ret) {
  if (/* invalid argument condition */) return QDMI_ERROR_INVALIDARGUMENT;
  ADD_STRING_PROPERTY(QDMI_DEVICE_PROPERTY_NAME, "Example Device", prop, size, value, size_ret)
  ADD_SINGLE_VALUE_PROPERTY(QDMI_DEVICE_PROPERTY_QUBITSNUM, size_t, 2, prop, size, value, size_ret)
  ADD_LIST_PROPERTY(QDMI_DEVICE_PROPERTY_COUPLINGMAP, EX_QDMI_Site,
      ((EX_QDMI_Site[]){SITES[0], SITES[1], SITES[1], SITES[0]}), 4, prop, size, value, size_ret)
  return QDMI_ERROR_NOTSUPPORTED;
}
```

Example Client

A QDMI device must implement all functions defined in the control and query interface. The QDMI driver loads all available devices and provides device handles to the QDMI clients. Those clients can use that device handle to create and submit a job, retrieve result data, or query information about the device. Following the querying of the device's coupling map from the client side is demonstrated.

```
/* get list of all device's sites */ size_t size = 0;
int ret = QDMI_device_query_device_property(device, QDMI_DEVICE_PROPERTY_SITES, 0, nullptr, &size);
throw_if_error(ret, "Failed to retrieve the sites list's size.");
std::vector<QDMI_Site> sites(size / sizeof(QDMI_Site));
ret = QDMI_device_query_device_property(device, QDMI_DEVICE_PROPERTY_SITES, size, sites.data(), nullptr);
throw_if_error(ret, "Failed to retrieve the sites list.");
/* query device's coupling map */ size = 0;
ret = QDMI_device_query_device_property(device, QDMI_DEVICE_PROPERTY_COUPLINGMAP, 0, nullptr, &size);
throw_if_error(ret, "Failed to query the coupling map's size.");
const auto num_pairs = size / sizeof(QDMI_Site) / 2;
std::vector<std::pair<QDMI_Site, QDMI_Site>> coupling_pairs(num_pairs);
ret = QDMI_device_query_device_property(device, QDMI_DEVICE_PROPERTY_COUPLINGMAP, size,
        coupling_pairs.data(), nullptr);
throw_if_error(ret, "Failed to query the coupling map.");
```

What is MQSS?

MQSS stands for Munich Quantum Software Stack, which is a project of the Munich Quantum Valley (MQV) initiative and is jointly developed by the Leibniz Supercomputing Centre (LRZ) and the Chairs for Design Automation (CDA), and for Computer Architecture and Parallel Systems (CAPS) at TUM. It provides a comprehensive compilation and runtime infrastructure for on-premise and remote quantum devices, support for modern compilation and optimization techniques, and enables both current and future high-level abstractions for quantum programming. Within the MQV, a concrete instance of the MQSS is deployed at the LRZ for the MQV, serving as a single access point to all of its quantum devices via multiple compatible access paths, including a web portal, command line access via web credentials as well as the option for hybrid access with tight integration with LRZ's HPC systems.

What is QDMI?

QDMI, or Quantum Device Management Interface, serves as the communication interface between software within the MQSS and the quantum hardware connected to the MQSS. The aim is to provide a standard way to communicate with quantum resources that can be widely used by the whole quantum community.

Where is the code?

The code is publicly available as open-source and hosted on GitHub at github.com/Munich-Quantum-Software-Stack/QDMI. An extensive documentation with examples, templates, and rationales can be found at munich-quantum-software-stack.github.io/QDMI.

Who is using QDMI?

QDMI will be the default communication channel within the MQSS, meaning all hardware and software tools integrated into the MQSS will have to support QDMI. Moreover, platforms implementing QDMI can also be seamlessly integrated in other software stacks understanding QDMI, as can software tools interfacing with QDMI for platform feedback.

Under which license is QDMI released?

QDMI is released under the Apache License v2.0 with LLVM Exceptions. Any contribution to the project is assumed to be under the same license.

Why is it written in C and not in Python?

The interface is written in C to allow close integration within the MQSS and fulfill the performance as well as stability requirements needed for production systems, in particular as we scale quantum systems. Further, this enables a clean integration into existing and well-established system software stacks, including those for HPC.

Can I still integrate my Python code?

Python natively allows calling C APIs. So while it might not be as straightforward as the usage from C/C++, it is definitely possible. However, we generally do expect Python-based programming approaches to be used as front-ends, feeding into a natively implemented compiler infrastructure, which then relies on QDMI. This is very similar to how Python is used in many other parts of high-performance computing.

Technical Contact
Lukas Burgholzer: lukas.burgholzer@tum.de
Yannick Stade: yannick.stade@tum.de

General Contact
Robert Wille, Martin Schulz, Jorge Echavarria
mqss@munich-quantum-valley.de