Ping Custom Op

Table of contents

List of Figures

In this section, we will modify the previous `ping_simple` example to add a custom operator into the workflow. We've already seen a custom operator defined in the `hello_world` example but skipped over some of the details.

In this example we will cover:

- the details of creating your own custom operator class
- how to add input and output ports to your operator
- how to add parameters to your operator
- the data type of the messages being passed between operators

> **Note**
>
> The example source code and run instructions can be found in the examples directory on GitHub, or under `/opt/nvidia/holoscan/examples` in the NGC container and the debian package, alongside their executables.

Operators and Workflow

Here is the diagram of the operators and workflow used in this example.

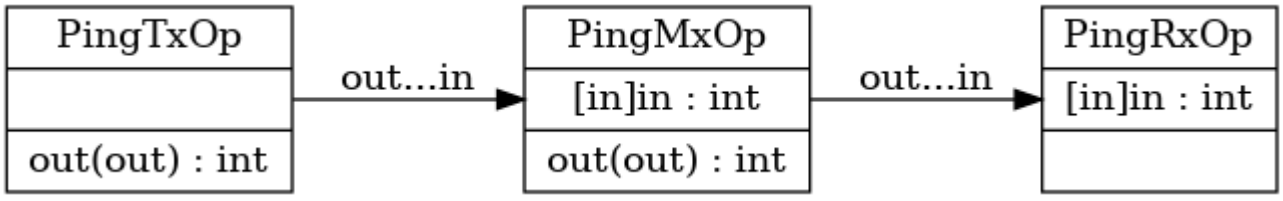

Fig. 6 *A linear workflow with new custom operator*

Compared to the previous example, we are adding a new **PingMxOp** operator between the **PingTxOp** and **PingRxOp** operators. This new operator takes as input an integer, multiplies it by a constant factor, and then sends the new value to **PingRxOp**. You can

think of this custom operator as doing some data processing on an input stream before sending the result to downstream operators.

Configuring Operator Input and Output Ports

Our custom operator needs 1 input and 1 output port and can be added by calling `spec.input()` and `spec.output()` methods within the operator's `setup()` method. This requires providing the data type and name of the port as arguments (for C++ API), or just the port name (for Python API). We will see an example of this in the code snippet below. For more details, see Specifying operator inputs and outputs (C++) or Specifying operator inputs and outputs (Python).

Configuring Operator Parameters

Operators can be made more reusable by customizing their parameters during initialization. The custom parameters can be provided either directly as arguments or accessed from the application's YAML configuration file. We will show how to use the former in this example to customize the "multiplier" factor of our **PingMxOp** custom operator. Configuring operators using a YAML configuration file will be shown in a subsequent example. For more details, see Configuring operator parameters.

The code snippet below shows how to define the **PingMxOp** class.

Ingested Tab Module

Now that the custom operator has been defined, we create the application, operators, and define the workflow.

Ingested Tab Module

Message Data Types

For the C++ API, the messages that are passed between the operators are the objects of the data type at the inputs and outputs, so the `value` variable from lines 20 and 25 of the example above has the type `int`. For the Python API, the messages passed between operators can be arbitrary Python objects so no special consideration is needed since it is not restricted to the stricter parameter typing used for C++ API operators.

Let's look at the code snippet for the built-in **PingTxOp** class and see if this helps to make it clearer.

Ingested Tab Module

> ⚠ **Attention**
>
> For advance use cases, e.g., when writing C++ applications where you need interoperability between C++ native and GXF operators you will need to use the `holoscan::TensorMap` type instead. See Interoperability between GXF and native C++ operators for more details. If you are writing a Python application which needs a mixture of Python wrapped C++ operators and native Python operators, see Interoperability between wrapped and native Python operators

Running the Application

Running the application should give you the following output in your terminal:

```
Middle message value: 1 Rx message value: 3 Middle message value: 2 Rx message value: 6 Middle message value: 3 Rx message value: 9 Middle message value: 4 Rx message value: 12 Middle message value: 5 Rx message value: 15 Middle message value: 6 Rx message value: 18 Middle message value: 7 Rx message value: 21 Middle message value: 8 Rx message value: 24 Middle message value: 9 Rx message value: 27 Middle message value: 10 Rx message value: 30
```