Using Holoscan Operators in GXF Applications

Table of contents

For users who are familiar with the GXF development ecosystem (used in Holoscan SDK 0.2), we provide an export feature to leverage native Holoscan operators as GXF codelets to execute in GXF applications and GraphComposer.

We demonstrate how to wrap a native C++ holoscan operator as a GXF codelet in the `wrap_operator_as_gxf_extension`, as described below.

1. Creating compatible Holoscan Operators

> **Note**
>
> This section assumes you are already familiar with how to create a native C++ operator.

To be compatible with GXF codelets, inputs and outputs specified in `Operator::setup(OperatorSpec& spec)` must be of type `holoscan::gxf::Entity`, as shown in the PingTxNativeOp and the PingRxNativeOp implementations of this example, in contrast to the PingTxOp and PingRxOp built-in operators of the SDK.

For more details regarding the use of `holoscan::gxf::Entity`, follow the documentation on Interoperability between GXF and native C++ operators.

2. Creating the GXF extension that wraps the operator

To wrap the native operator as a GXF codelet in a GXF extension, we provide the CMake `wrap_operator_as_gxf_extension` function in the SDK. An example of how it wraps `PingTxNativeOp` and `PingRxNativeOp` can be found here.

- It leverages the CMake target names of the operators defined in their respective `CMakeLists.txt` (ping_tx_native_op, ping_rx_native_op)
- The function parameters are documented at the top of the WrapOperatorAsGXFExtension.cmake file (ignore implementation below).

> ⚠ **Warning**
>
> - A unique GXF extension is currently needed for each native operator to export (operators cannot be bundled in a single extension at this time).
> - Wrapping other GXF entities than operators (as codelets) is not currently supported.

3. Using your wrapped operator in a GXF application

> ⓘ **Note**
>
> This section assumes you are familiar with how to create a GXF application.

As shown in the `gxf_app/CMakeLists.txt` here, you need to list the following extensions in `create_gxe_application()` to use your wrapped codelets:

- `GXF::std`
- `gxf_holoscan_wrapper`
- the name of the CMake target for the created extension, defined by the `EXTENSION_TARGET_NAME` argument passed to `wrap_operator_as_gxf_extension` in the previous section

The codelet class name (defined by the `CODELET_NAMESPACE::CODELET_NAME` arguments passed to `wrap_operator_as_gxf_extension` in the previous section) can then be used as a component `type` in a GXF app node, as shown in the YAML app definition of the example, connecting the two ping operators.