Conditions

Table of contents

The following table shows various states of the scheduling status of an operator:

Scheduling Status	Description
NEVER	Operator will never execute again
READY	Operator is ready for execution
WAIT	Operator may execute in the future
WAIT_TIME	Operator will be ready for execution after specified duration
WAIT_EVENT	Operator is waiting on an asynchronous event with unknown time interval

Note

- A failure in execution of any single operator stops the execution of all the operators.
- Operators are naturally unscheduled from execution when their scheduling status reaches `NEVER` state.

By default, operators are always `READY`, meaning they are scheduled to continuously execute their `compute()` method. To change that behavior, some condition classes can be passed to the constructor of an operator. There are various conditions currently supported in the Holoscan SDK:

- MessageAvailableCondition
- DownstreamMessageAffordableCondition
- CountCondition
- BooleanCondition
- PeriodicCondition

- AsynchronousCondition

> **Note**
>
> Detailed APIs can be found here: C++/
> `<a href="../api/python/holoscan_python_api_conditions.html#module-holoscan.conditions">Python</a>`
>).

Conditions are AND-combined

An Operator can be associated with multiple conditions which define its execution behavior. Conditions are AND combined to describe the current state of an operator. For an operator to be executed by the scheduler, all the conditions must be in `READY` state and conversely, the operator is unscheduled from execution whenever any one of the scheduling terms reaches `NEVER` state. The priority of various states during AND combine follows the order `NEVER`, `WAIT_EVENT`, `WAIT`, `WAIT_TIME`, and `READY`.

MessageAvailableCondition

An operator associated with `MessageAvailableCondition` (`C++` / `Python`) is executed when the associated queue of the input port has at least a certain number of elements. This condition is associated with a specific input port of an operator through the `condition()` method on the return value (IOSpec) of the OperatorSpec's `input()` method.

The minimum number of messages that permits the execution of the operator is specified by `min_size` parameter (default: `1`). An optional parameter for this condition is `front_stage_max_size`, the maximum front stage message count. If this parameter is set, the condition will only allow execution if the number of messages in the queue does not exceed this count. It can be used for operators which do not consume all messages from the queue.

DownstreamMessageAffordableCondition

The `DownstreamMessageAffordableCondition` (`C++` / `Python`) condition specifies that an operator shall be executed if the input port of the downstream operator for a given output port can accept new messages. This condition is associated with a specific output port of an operator through the `condition()` method on the return value (IOSpec) of the OperatorSpec's `output()` method. The minimum number of messages that permits the execution of the operator is specified by `min_size` parameter (default: `1`).

CountCondition

An operator associated with `CountCondition` (`C++` / `Python`) is executed for a specific number of times specified using its `count` parameter. The scheduling status of the operator associated with this condition can either be in `READY` or `NEVER` state. The scheduling status reaches the `NEVER` state when the operator has been executed `count` number of times.

BooleanCondition

An operator associated with `BooleanCondition` (`C++` / `Python`) is executed when the associated boolean variable is set to `true` . The boolean variable is set to `true` / `false` by calling the `enable_tick()` / `disable_tick()` methods on the `BooleanCondition` object. The `check_tick_enabled()` method can be used to check if the boolean variable is set to `true` / `false` . The scheduling status of the operator associated with this condition can either be in `READY` or `NEVER` state. If the boolean variable is set to `true` , the scheduling status of the operator associated with this condition is set to `READY` . If the boolean variable is set to `false` , the scheduling status of the operator associated with this condition is set to `NEVER` . The `enable_tick()` / `disable_tick()` methods can be called from any operator in the workflow.

Ingested Tab Module

PeriodicCondition

An operator associated with `PeriodicCondition` (`C++` / `Python`) is executed after periodic time intervals specified using its `recess_period` parameter. The scheduling status of the operator associated with this condition can either be in `READY` or `WAIT_TIME` state. For the first time or after periodic time intervals, the scheduling status

of the operator associated with this condition is set to `READY` and the operator is executed. After the operator is executed, the scheduling status is set to `WAIT_TIME`, and the operator is not executed until the `recess_period` time interval.

AsynchronousCondition

`AsynchronousCondition` (`C++` / `Python`) is primarily associated with operators which are working with asynchronous events happening outside of their regular execution performed by the scheduler. Since these events are non-periodic in nature, `AsynchronousCondition` prevents the scheduler from polling the operator for its status regularly and reduces CPU utilization. The scheduling status of the operator associated with this condition can either be in `READY`, `WAIT`, `WAIT_EVENT`, or `NEVER` states based on the asynchronous event it's waiting on.

The state of an asynchronous event is described using `AsynchronousEventState` and is updated using the `event_state()` API.

AsynchronousEventState	Description
READY	Init state, first execution of `compute()` method is pending
WAIT	Request to async service yet to be sent, nothing to do but wait
EVENT_WAITING	Request sent to an async service, pending event done notification
EVENT_DONE	Event done notification received, operator ready to be ticked
EVENT_NEVER	Operator does not want to be executed again, end of execution

Operators associated with this scheduling term most likely have an asynchronous thread which can update the state of the condition outside of its regular execution cycle performed by the scheduler. When the asynchronous event state is in `WAIT` state, the scheduler regularly polls for the scheduling state of the operator. When the asynchronous event state is in `EVENT_WAITING` state, schedulers will not check the scheduling status of the operator again until they receive an event notification. Setting the state of the asynchronous event to `EVENT_DONE` automatically sends the event notification to the scheduler. Operators can use the `EVENT_NEVER` state to indicate the

end of its execution cycle. As for all of the condition types, the condition type can be used with any of the schedulers.