Cog Sensors and Actuators

External Reference Specification

Version 2

Greg Snider

# Overview

Cog Sensors and Actuators are the inputs and outputs of Compute Graphs, exchanging information between an application running on the CPU and the Compute Graph computation which may be running across multiple multi-core engines. Sensors and Actuators are typically native components that link with the Native Runtime. However, their interface is simple enough that they may be written in any language that can link with C structs and functions, and with the Native Runtime library.

Sensors and actuators reside in the Cog application and are “plugged in” to the Runtime as a part of the Runtime initialization. They are invoked by the Runtime through callbacks:

Cog App

Cog Native Runtime

GPU

GPU

GPU

GPU

actuator

sensor

callbacks

control

# Interface

Sensors and actuators have nearly identical interfaces. Each contains a major and minor version number to ensure compatibility with the Compute Graph and Native Runtime. Each contains a description of the tensor field that they hold. And each defines three callback functions, called by the Runtime, to implement the sensor or actuator functionality.

#### Sensor

A sensor is defined with the following struct:

struct CogSensor {

int majorVersionNumber;

int minorVersionNumber;

struct CogTensorField fieldDescriptor;

void (\*resetSensor)(struct CogSensor\* sensor, float\* buffer);

void (\*writeSensor)(struct CogSensor\* sensor, float\* buffer);

void (\*closeSensor)(struct CogSensor\* sensor);

}

The resetSensor() function is called by the Runtime whenever the ComputeGraph is reset by the application. This callback function initializes the sensor (for example: opening a file for reading, initializing a camera) and fills buffer with initial data.

The writeSensor() function is called by the Runtime whenever the ComputeGraph is ready to read the next tensor field from the input sensor stream. This callback function fills buffer with the next field of data.

The closeSensor() function is called whenever the application calls shutdown(). This gives the sensor the chance to release resources (for example: flushing and closing open files, closing devices) in an orderly manner.

Note that all buffers passed to the callback functions are owned by the Runtime and may not be reused by the application for anything else.

#### Actuator

An actuator is defined with the following struct:

struct CogActuator {

int majorVersionNumber;

int minorVersionNumber;

struct CogTensorField fieldDescriptor;

void (\*resetActuator)(struct CogActuator\* actuator, float\* buffer);

void (\*readActuator)(struct CogActuator\* actuator, float\* buffer);

void (\*closeActuator)(struct CogActuator\* actuator);

}

The resetActuator() function is called by the Runtime whenever the ComputeGraph is reset by the application. This callback function initializes the actuator (for example: opening a file for writing, initializing a display or microphone) and fills buffer with initial data.

The readActuator() function is called by the Runtime whenever the ComputeGraph is ready to read the next tensor field from the input sensor stream. This callback function reads buffer to extract the sensor data.

The closeActuator() function is called whenever the application calls shutdown(). This gives the actuator the chance to release resources (for example: flushing and closing files, closing devices) in an orderly manner.

Note that all buffers passed to the callback functions are owned by the Runtime and may not be reused by the application for anything else.

# Reading and Writing Buffers

Buffers in callback functions hold tensor fields. The layout for those fields is defined in the Cog Tensor Field External Reference Specification. That document also defines the CogTensorField struct.

# Installing Sensors and Actuators

The Runtime supplies functions for installing sensors and actuators. See the Cog Native Runtime External Reference Specification for details.

# Standard Sensors

Cog is supplied with four standard, native sensors:

struct CogSensor createMovieSensor(char\* filename):

A sensor that reads a movie file and writes it one frame at a time on each call of the writeSensor() callback. The first frame of the movie is written by the resetSensor() callback. The movie automatically loops back to the first frame after the last frame is read. The closeSensor() callback closes the movie file.

struct CogSensor createCameraSensor(char\* deviceName):

A sensor that reads a video camera and writes it one frame at a time on each call of the writeSensor() callback. The resetSensor() callback supplies an initial image from the camera. The closeSensor() callback closes the camera.

struct CogSensor createImageSensor(char\* filename):

A sensor that opens an image file, writes it to the sensor buffer during the resetSensor() callback, then closes it. The writeSensor() and closeSensor() callbacks do nothing.

struct CogSensor createBinarySensor(char\* hdf5Filename, char\* hdf5Path):

A sensor that opens an hdf5 binary file, seeks to a path within the file specified by hdf5Path, and extracts and writes a stream of field data to the sensor buffer during the resetSensor() callbacks. This is used to emulate sensors other than movies, cameras and images.

These are visible in Scala as function objects:

MovieSensor(filename: String): Sensor

CameraSensor(devicename: String): Sensor

ImageSensor(filename: String): Sensor

BinarySensor(hdf5Filename: String, hdf5Path: String): Sensor

# Standard Actuator

Cog is supplied with a single native actuator:

struct CogActuator createBinaryActuator(char\* hdf5Filename, char\* hdf5Path):

An actuator that opens an hdf5 binary file, seeks to a path within the file specified by hdf5Path, and writes a stream of field data to the sensor buffer during the readActuator() callbacks.

This is visible in Scala as a function object:

BinaryActuator(hdf5Filename: String, hdf5Path: String): Sensor

# Debugging Cog applications

Debugging Cog applications is usually easier when sensor streams are controlled and deterministic. Hence we suggest using one of the standard sensors when developing a Compute Graph. Unique sensors can be emulated using BinarySensor. When deployed, the application can install its own native sensors to replace the built-in sensor stubs (the Runtime allows previously installed sensors and actuators to be replaced with new actuators and sensors).