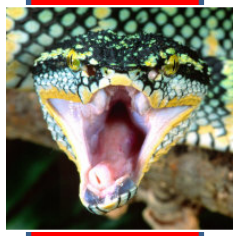
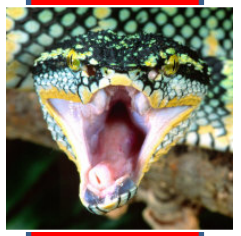


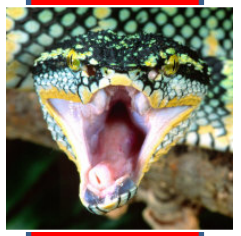
OpenCPI's Application Control Interface (ACI)

Agenda



- Introduction
- Creating a New Application
- A Simple Example
- Compiling ACI Applications
- Control Plane
- Accessing the Data Plane
- Python Interface

Introduction

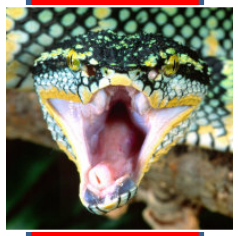


- The ACI is an API for executing XML-based OpenCPI applications within a C++ or Python context

- Useful when '`ocpirun`' cannot supply the necessary programmatic and/or dynamic creation or control of an XML-based application
  - The contents of the application XML (OAS) need to be constructed programmatically
  - The main program needs to directly connect to the ports of the running application
  - An XML-based application needs to be run repeatedly in the same process
  - Some attributes or properties of the XML application need to be dynamically configured or read throughout execution of the application

- Useful for standalone applications or test fixtures

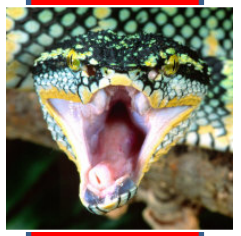
Creating a New Application



- ACI applications are placed under the applications/ subdirectory of an OpenCPI project
- New ACI applications can be created with the ocpidev tool as follows:
  - `$ ocpidev create application ${appname}`
- If `${appname}` was `demo-aci-app`, the following directory and files should be created as follows:

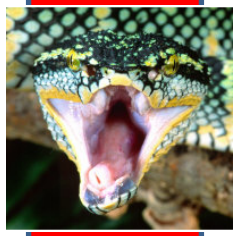
```
applications/
|-- demo-aci-app/
|   |-- demo-aci-app.cc
|   |-- demo-aci-app.xml
|   `-- Makefile
`-- Makefile
```

A Simple Example



```
#include “OcpiApi.hh”
namespace OA = OCPI::API;
...
int main()
{
  try {
    OA::Application app(“demo-aci-app.xml”);
    app.initialize();
    app.start();
    app.wait();
    app.finish();
  catch (std::string &e) {
    std::cerr << “Error: “ << e << std::endl;
  }
}
```

Compiling ACI Applications



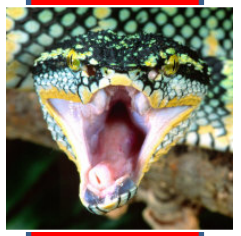
- Compiling an ACI application is similar to building other OpenCPI assets

- The '--rcc-platform' option can be used to build for various software platforms

- From the lower level directory:
  $ ocpidev build --rcc-platform centos7 --rcc-platform xilinx13_3

- From top-level of directory:
  $ ocpidev build application demo-aci-app --rcc-platform xilinx13_3

Using the Control Plane



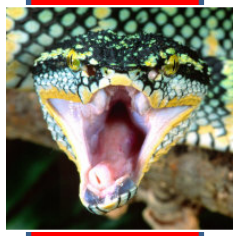
- Control plane operations in OpenCPI consist of getting and setting properties of components

- In the ACI, there are a few different options for getting and setting properties depending on performance, accuracy, and simplicity requirements

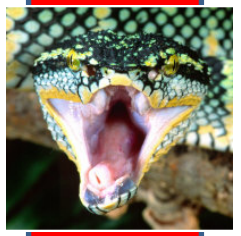
Getting Properties – Option 1

```
...
    OA::Application app("demo-aci-app.xml");
    app.initialize();
    std::string value;
    app.getProperty("bias", "biasValue", value);
    std::cout << value << "\n";
...
    app.start();
    while (true) {
      ...
    }
  catch (std::string &e) {
    std::cerr << "Error: " << e << std::endl;
  }
}
```

- Simplest interface but all string-based, so slowest for real-time processing.



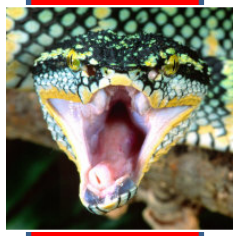
Getting Properties – Option 2



```
...
    OA::Application app("demo-aci-app.xml");
    app.initialize();
    unsigned int ivalue = app.getPropertyValue("bias", "biasValue");
    std::cout << ivalue << "\n";
...
    app.start();
    while (true) {
      ...
    }
  catch (std::string &e) {
    std::cerr << "Error: " << e << std::endl;
  }
}
```

- Template based and simple, but can have C++ 'promoting' issues if you don't explicitly state the type using `getPropertyValue<type>()`. Gives proper native types, which is sometimes more convenient.

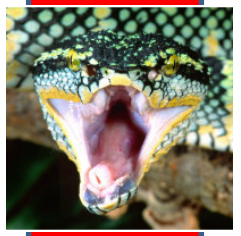
Getting Properties – Option 3



```
...
    OA::Application app("demo-aci-app.xml");
    app.initialize();
    OA::Property biasValue(app, "bias.biasValue");
    std::cout << biasValue.getULongValue();
...
    app.start();
    while (true) {
      ...
    }
  catch (std::string &e) {
    std::cerr << "Error: " << e << std::endl;
  }
}
```

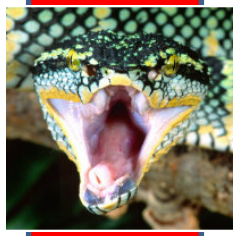
- Fastest, lowest level, "I know exactly what to expect," with possible optimizations to direct memory-mapped access.

Setting Properties



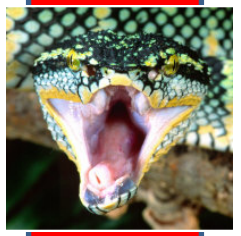
```
…
   OA::Application app(“myapp.xml”);
   app.initialize();
   app.setProperty(“bias”, “biasValue”, “15”);

   app.setPropertyValue<unsigned int>(“bias”, “biasValue”, 16);

   OA::Property biasValue(app, “bias.biasValue”);
   biasValue.setULongValue(17);
…
   app.start();
   while (true) {
     …
   }
  catch (std::string &e) {
   std::cerr << “Error: “ << e << std::endl;
  }
}
```

Accessing the Data Plane



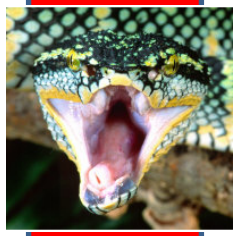
- Data can be written into or read from External Ports of an OpenCPI application using the ACI's ExternalPort and ExternalBuffer classes
- Useful for interfacing with other applications external to OpenCPI that use a different protocol or transport layer
- Leaves the data formatting and opcode up to the user for increase flexibility

Accessing the Data Plane



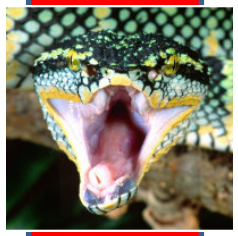
```
#include “OcpiApi.hh”
namespace OA = OCPI::API;
...
int main()
{
  try {
    OA::Application app(“demo-aci-app.xml”);
    app.initialize();
    OA::ExternalPort &toApp = app.getPort(“in”);
    OA::ExternalPort &fromApp = app.getPort(“out”);
    app.start();
    while (true) {
      ...
    }
  catch (std::string &e) {
    std::cerr << “Error: “ << e << std::endl;
  }
}
```

Example of Sending Data *to* an Application



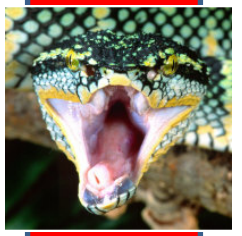
```
size_t sendBytesToApp(uint8_t *sendPtr, size_t sendSize, ExternalPort &port)
{
  uint8_t *dataPtr = NULL;
  size_t dataSizeBytes = 0;
  OA::ExternalBuffer *dataBuffer = NULL;

  // Get a buffer to fill
  do {
    dataBuffer = port.getBuffer(dataPtr, dataSizeBytes);
  } while(!dataBuffer);

  // Fill the buffer
  const size_t transferSizeBytes = std::min(sendSize, dataSizeBytes);
  memcpy(dataPtr, sendPtr, transferSizeBytes);

  // Actually push N bytes on the port with opcode 0
  dataBuffer->put(transferSizeBytes, 0);
  return transferSizeBytes; // Tell caller
}
```

Example of Getting Data *from* an Application



```
size_t getBytesFromApp(uint8_t *recvPtr, size_t recvSize, ExternalPort &port)
{
  uint8_t *dataPtr = NULL;
  size_t dataSizeBytes = 0;
  uint8_t opCode;
  bool endOfData;
  OA::ExternalBuffer *dataBuffer = NULL;

  // Get a buffer of data
  do {
    dataBuffer = port.getBuffer(dataPtr, dataSizeBytes, opCode, endOfData);
  } while(!dataBuffer);

  // Fill the out buffer with the data
  const size_t transferSizeBytes = std::min(recvSize, dataSizeBytes);
  memcpy(recvPtr, dataPtr, transferSizeBytes);

  // Release the buffer
  dataBuffer->release();
  return transferSizeBytes; // Tell caller
}
```

SWIG Python Interface (Beta)



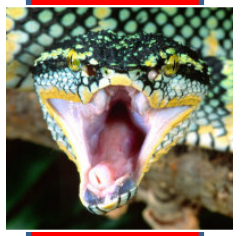
- OpenCPI has released SWIG bindings for the ACI which allows users to access the ACI from within a python interpreter

- Once installed, the ACI can be imported into a python script:

```
$ python
>>> import OcpiApi as OA # 1.3
>>> import opencpi.aci as OA # 1.4
```

- It is mostly a 1:1 translation from C++ ACI functions to python
  - There are some exceptions due to differences in language features

A Simple Python Example



```
import opencpi.aci as OA
try:
    app = OA.Application("demo-aci-app.xml")
    app.initialize()
    app.start()
    app.wait()
except Exception as e:
   print "Error: " + str(e)
```