

DataCutter 3 Tutorial

Christian Hansen

University of Maryland

(chansen@cs.umd.edu)

Introduction

- Programming tasks
 - Filter, filter library creation
 - Console application creation
- Compilation
- Setting up your environment
- Running DataCutter and your code

Filter Specification

- Define filters as a child of the DCFilterImplementation class
 - Need a single ‘process’ function that will be called for each unit of work

```
#include <DataCutterFilterDeveloper.h>
class Reader: public DCFilterImplementation {
public:
    DC_RTN_t process(DCFilterArg &arg);
private:
    DCBuffer buf;
};
```

Library Specification

- Define a library containing the filters
 - Library constructor registers specification objects for filters contained in the library

```
class FileOps: public DCFilterLibrary {
public:
    FileOps(): DCFilterLibrary("FileOPS") {
        this->registerFilter(new
        DCFilterSpec("Reader", DC_CreateFilter<Reader>,
        1, 1));
    }
};
```

Library Implementation

- Define a `dcLoadFilterFactory` function that will be called when the library is loaded by the appds

```
extern "C"  
DCFilterLibrary* dcLoadFilterFactory() {  
    return new FileOps();  
}
```

Filter Implementation

- Retrieve the work definition from DCFilterArg

```
DC_RTN_t Reader::process(DCFilterArg &arg) {  
    char* msg = arg.pwork->buf.getPtr();  
    cout << "Retrieving file " << msg << "... ";  
  
    ifstream ifs;  
    ifs.open(msg, ios::binary);  
    if (!ifs.is_open()) {  
        return DC_RTN_ABRT;  
    }  
}
```

Filter Implementation (cont)

- Allocate buffer space
- Write to the output stream

```
buf.New(8192);
while (ifs.read(buf.getPtr(), 8192)) {
    buf.setSize(8192);
    if (arg.outs[0].write(&buf) != DC_ERR_OK) {
        return DC_RTN_ABRT;
    }
    buf.setSize(0);
}
```

Filter Implementation (cont)

- Write the last buffer and return the status
- Be sure to indicate how much of the buffer is being used

```
if (ifs.gcount() > 0) {  
    buf.setSize(ifs.gcount());  
    if (arg.outs[0].write(&buf) != DC_ERR_OK) {  
        return DC_RTN_ABRT;  
    }  
}  
ifs.close();  
return DC_RTN_OK;  
}
```

Console Implementation

- The “console” drives the application
 - Initialize DataCutter and load filter libraries
 - Specify the filter connectivity, i.e. layout
 - Map filters to specific hosts, i.e. placement
 - Instantiate filters
 - Create and assign work definitions
 - Communicate with client application

Console Implementation (cont)

- Initialize DataCutter and load filter libraries

```
#include <DataCutterApplicationDeveloper.h>
int main(int argc, char* argv[]) {
    DCInstanceFactory* dcif = DCInstanceFactory::New();
    DCFilterFactory* dcff = new DCFilterFactory();
    if (dcif->Init("dccopy", dcff, &argc, &argv)) {
        return 1;
    }
    if (dcff->registerLibrary("fileops.so") !=
        DC_ERR_OK) {
        return 1;
    }
}
```

Console Implementation (cont)

- Create filter objects
- The “console” can behave as a filter

```
DCFilter* reader = dcff->createFilter("reader",
    "Reader");
DCFilter* writer = dcff->createFilter("writer",
    "<console>");
if (!reader || !writer) {
    return 1;
}
```

Console Implementation (cont)

- Create a layout by connecting filter input ports to output ports

```
DCLayout* layout = new DCLayout();  
layout->addFilter(reader);  
layout->addFilter(writer);  
layout->addPipe(DCPipe::create(reader->  
>getOutPort(0), writer->getInPort(0)));
```

Console Implementation (cont)

- Pin filters to specific hosts using a placement

```
DCResources* dcrs = DCResources::getInstance();
DCPlacement* placement = new DCPlacement(layout);
if (placement->add(reader, dcrs->
    >getHost(DCString_t(argv[1]))) != DC_ERR_OK) {
    return 1;
}
```

Console Implementation (cont)

- Instantiate the set of filters given in the placement (or layout)

```
int sts;
DCInstance* instance;
if ((sts = dcif->NewInstance(instance, placement)) !=
    DC_ERR_OK) {
    return 1;
}
```

Console Implementation (cont)

- Create the work definition and assign to an instance

```
DCWork work;  
work.buf.Empty();  
work.buf.Append(argv[2], strlen(argv[2])+1);  
  
int wh;  
if ((wh = instance->AppendWork(work)) < 0) {  
    return 1;  
}
```

Console Implementation (cont)

- Use the instance to access a stream if the console is part of the layout

```
ofstream ofs;
ofs.open(argv[3], ios::binary);
if (!ofs.is_open()) {
    return 1;
}
DCBuffer* pbuf;
while ((pbuf = instance->arg.ins[0].read())) {
    ofs.write(pbuf->getPtr(), pbuf->getSize());
    pbuf->consume();
}
ofs.close();
```

Console Implementation (cont)

- Await work completion and check the filter exit status

```
sts = dcif->WaitWork(wh);  
if (sts == DC_ERR_RemoteFilterFail || sts ==  
    DC_ERR_InstanceFail) {  
    cout << "filter error" << endl;  
}
```

Console Implementation (cont)

- Clean up after yourself

```
dcif->StopInstance(instance);
```

```
delete placement;
```

```
delete layout;
```

```
delete dcff;
```

```
delete dcif;
```

```
return 0;
```

```
}
```

Filter Library Compilation

- Include DataCutterFilterDeveloper.h
- See examples/Makefile for the required defines, includes, and libraries
- Make the library shared
 - Compile code with ‘-fPIC’
 - Link object files with ‘-shared’

Console Compilation

- Include `DataCutterApplicationDeveloper.h`
- See `examples/Makefile` for the required defines, includes, and libraries
- Add `/roguehomes/chansen/work/818-DataCutter/lib` to your `LD_LIBRARY_PATH` environment variable

Setting the Environment

- Add the run-time binaries to your path
 - set path = \$path:/roguehomes/chansen/work/818-DataCutter/bin
- set environment variables
DATACUTTER_DIRDHOST and
DATACUTTER_DIRDPORT
- Generate authentication keys for ssh
 - man ssh-keygen

Starting the Run-Time System

- Start dird
 - `bootUtil dird start`
- Start appds
 - `bootUtil appd start red01 red02 blue01 ...`
- Trouble?
 - Can you ssh into the remote host?
 - Can you start dird/appd manually?

Starting your application

- Put your filter library in a directory common to all hosts
 - Just use your home directory on redleader
 - Add its location to your `LD_LIBRARY_PATH`
- Run your application and watch the output
 - Filter output will appear in small grey terminal windows created by `bootUtil`
- Do not read/write data from/to NFS directories!
 - except for input data read filters