

IF Data Handling Layer Design

Overview

The data handling layer for the Intensity Frontier experiments needs to meet the various requirements in the Data Handling Requirements for Intensity Frontier Experiments. This document presents a design for tools to assist in that. As each component is discussed, the requirements it meets will be reviewed.

Rationale

We want to provide an easy migration path for IF experiments who currently operate in a mode where they run site-local batch jobs where their experiment data disks are visible via NFS and all of their data is on disk, to a mode where they run batch jobs OSG-wide, and possibly stage to and from tape via intermediate disk with SAM. They should be able to do this in stages, where they:

- replace **cp** or **cpn** commands with **ifdh** commands;
- declare metadata for their files into SAM;
- use metadata queries to build data-sets for analysis
- use metadata/dataset project queries to get file locations

without interrupting or significantly changing their current work-flow.

Expected Usage

We expect users to

- declare datasets for their jobs to analyze, either with the web GUI or with `ifdh define_dataset`
- submit jobs with our DAG based jobs submit script
- in those jobs, obtain input files with a “`ifdh next_file`” call
- copy output files back to a staging area with “`ifdh cp`”
- Use the File Transfer Service to stage the output files to tape and/or semi-permanent disk storage

Packaging

The later versions of the data handling layer will be provided as two files: a small executable called **ifdh** and a shared library **ifdh.so** which will have the C++ and Python callable interfaces. Having it be only two files will facilitate it being transferred to Grid jobs so that they can be used to transfer other files. The library will use an https interface to contact a web service fronting the data handling service (i.e. SAM), to avoid the network firewall issues with the current SAM/Corba interface. Also, this simplified set of commands/functions should be sufficient for users submitting jobs, and be easier to learn than the full SAM suite of commands. Secondly, these calls could be implemented in many cases without SAM, but by some other set of data handling tools.

Early (prototype) versions of the package may have only the command line tool implemented as a shell or python script.

Functional Specification

There will be several areas of functionality in this executable and library, described below. Each of these will be defined in terms of method calls to an ifdh object, but they will also be provided by the ifdh executable; so that

```
ifdh.method(arg, arg,...)
```

in a C++ or Python program is equivalent of doing

```
ifdh method arg arg ...
```

in a shell script. The command line wrapper will present results of function calls that return (lists of) strings by printing results to stdout. The command line tool will exit with an exit code of the return value of functions returning an integer status.

Logging

int ifdh.log(*message*)

log message to IF job logging service Also, all of the other commands/calls below will log suitable information to the IF job logging service. A zero return indicates success.

File Transfer

int ifdh.cp(*source/filename,destination*)

This will copy a file from a known source to the local disk or vice versa, and make sure any underlying data handling system is made aware. It will use CPN or equivalent for copies from visible NFS mounted storage, or use gridftp to obtain files not visible via NFS. A zero return value indicates success.

ifdh.rm(*location/filename*)

This will remove a file from a given location, updating any underlying data handling system. A zero return value indicates success.

Metadata/Datasets

In this scenario, jobs do not deal with metadata directly; the File Transfer Service stores files with metadata, and jobs do not themselves declare metadata to the data handing service. They are expected to put metadata into files (ART framework) or next to files.

(i.e. file.metadata.py) However, outside of jobs, scripts may need to do some metadata related work. So in a later release, the following calls will simply pass through to the SAM web interface.

int ifdh.createDefinition(*name, dimensionstring, username*)

int ifdh.deleteDefinition(*name*)

string ifdh.describeDefintion(*name*)

string ifdh.translateConstraints(*dimensionstring*)

string ifdh.locateFile(*filename*)

Also in a later release we may support the following to allow automated job submeission tools:

ifdh.recoveryFileset(*projectname, datasetname*)

Makes a fileset of files not completed in project *projectname*.

Projects

string ifdh.startProject(*projectname, station, datasetname, userstring [,group]*)

Starts a project to deliver the files in *filesetname* in some order. Returns a URL handle for the project.

string ifdh.establishConsumer(*projectname, appname, appversion, deliverylocation, user*)

Joins a project as a conusmer process, returns your project consumer url string.

string ifdh.nextFile(*consumer_url*)

Returns the location of the next file in the project, suitable for use with `ifdh.cp()`.

ifdh.releaseFile(*consumer_url, filename, status_ok*)

Declares you've finished reading the file, and whether it went ok.

int ifdh.endProcess(*consmer_url, int status_ok*)

Indicate you're done working on a project, and whether your handling of all files went ok.

Sample Job wrapper script

```
# values from invocation
inputfilename=...
outputfilename=...
projectname=...

# join project, get consumer process url
cpurl=`ifdh establishConsumer $projectname`

# say we're done on exit or kill signal
trap "ifdh endProcess $cpurl failed" 0 1 2 3 4 5 6 11 15
while :
do
    fname=`ifdh.nextFile $cpurl`
    if [ "x$fname" = "x" ]
    then
        # no more files...
        break
    fi

    ifdh cp $fname $inputfilename

    # process it
    executable arg1 arg2 ...

    # report processing it
    ifdh releaseFile $cpurl $fname ok

    # copy output file to staging area for FTS to file
    ifdh cp $outputfile /some/stage/area

    # if we have an external metadata file, copy it out , too
    ifdh cp $outputfile.metadata /some/stage/area

done
ifdh endProcess $cpurl
ifdh setstatus $cpurl ok
```