Data Server Interface Specification

Introduction

The purpose of this specification is to provide an “ICAT friendly” interface to a data server. It has been designed to meet the needs of Diamond, ISIS and LSF but it is anticipated that it will find more general acceptance and so avoid the need for specific TopCat plugins for different file servers (or download managers). It is planned to provide a simple reference implementation. The interface includes calls to put an individual datafile and calls to get, query the status of, and delete groups of datafiles as specified by the ids of investigations, datasets and datafiles. The specification does not define how the group of files will be returned but a recommendation for a zip file format is included which will be used in the reference implementation.

ICAT Coupling

ICAT session ids are passed as an argument to most calls. The implementation should check for **R**ead access to the referenced data by a suitable call to ICAT. For put and delete operations the implementation must check for **C**reate and **D**elete access to the referenced data and should make the corresponding changes to ICAT. The implementation must be coded to maintain consistency with ICAT. In the case of a failure of software or hardware it is preferable to have an orphan file rather than an entry in ICAT for which no file exists.

Web service style

The web service is defined following some of the “REST” guidelines. Specifically the HTTP methods PUT, DELETE, POST and GET will be used as appropriate. In addition, because the GET URLs may become too long, POST will be supported as an alternative to GET when retrieving datafiles. To avoid the cost of encoding the datafile metadata when uploading a datafile the metadata are transmitted as request properties and so travel in the http header. A number of the calls take parameters: investigationIds, datasetIds and datafileIds; each of these take the form of a comma separated list.

Dual storage model

The assumed model is one where datafiles are held as far as possible on fast storage. The implementation is free to move datafiles to slower storage, thereby freeing up space on the fast storage as necessary. There is no necessity for an implementation to introduce this complexity if sufficient fast storage should be available. An implementation may choose to treat the fast storage as a non-backuped up cache of the slower, backup-up, storage.

The calls to be implemented

put

http method: PUT
request header fields: sessionId, name, location, description, fileSize, doi, checksum, datafileCreateTime, datafileModTime, datafileFormatId, datasetId
return: string representation of the id of the created datafile

- The body of the servlet request is the data of the file to be stored
- datafileModTime and datafileCreateTime must be in the format YYYY-MM-DD hh:mm:ss
- **C**reate permission is required

delete

http method: DELETE
parameters: sessionId, investigationIds, datasetIds, datafileIds

- Deletion of investigations, datasets and datafiles causes the contained components to be deleted both from the datasetver and from ICAT
- **D**elete permission is required

archive

http method: POST
parameters: sessionId, investigationIds, datasetIds, datafileIds

- Archiving of investigations, datasets and datafiles is a hint that the datafiles may be moved to storage where access may be slower
- **R**ead permission is required. This is because there is no risk to the data, an archiving request can at most delay access.

restore

http method: POST
parameters: sessionId, investigationIds, datasetIds, datafileIds

- Restoration of investigations, datasets and datafiles is a hint that the datafiles be moved to storage where access is faster
- **R**ead permission is required.

getStatus

http method: GET
parameters: investigationIds, datasetIds, datafileIds
return: a string with “ONLINE” if all data are available of fast storage, “RESTORING” if some data has been requested to move to fast storage and any other datafiles are already on fast storage or “ARCHIVED”.

- This does not require a sessionId so no permissions are required.

getData

http method: GET
parameters: sessionId, investigationIds, datasetIds, datafileIds, compress
return: the requested datafile or datafiles.

- If some or all of the data are not on fast storage the implementation should commence retrieval of the data and attach an exit code to the response.
- compress may have the value 1 to indicate a high degree of compression, 0 to indicate none and if omitted the level of compression is implementation defined.

Return codes

- SC_OK (200) for most successful calls - but see next one.
- SC_CREATED (201) for a successful put.
- SC_BAD_REQUEST (400) for bad or missing parameters.
- SC_FORBIDDEN (403) for authn or authz errors.
- SC_NOT_FOUND (404) if one or more of the requested items is not recognised.

Format of the content returned by getData

This is a recommendation and is not required by the specification as it is understood that some facilities may wish to offer a familiar view of the data. It is based on a simple hierarchical structure of datafiles within datasets within investigations.

1. If one datafile is requested it is returned without a container unless compress = 1 has been specified. The name of the file should be “Datafile-<ID>” where ID is the requested id value.
2. In cases other than shown above in 1 a zip file should be used. The zip file should have a name derived from the date and time it was generated. This should have the form YYYY-MM-DD-hh:mm:ss.zip. Note the “-” between the date and time to make the name more unix friendly.
3. Top level entries in the zip file should have names of the form “Investigation-<ID>”, “Dataset-<ID>” or “Datafile-<ID>” where ID is the requested id value. Second level, if present, should be of the form “Dataset-<ID>” for the dataset directory within an investigation or “Datafile-<ID>” for the datafile within a dataset. Third level, if present, will be a file name of the form “Datafile-<ID>”.