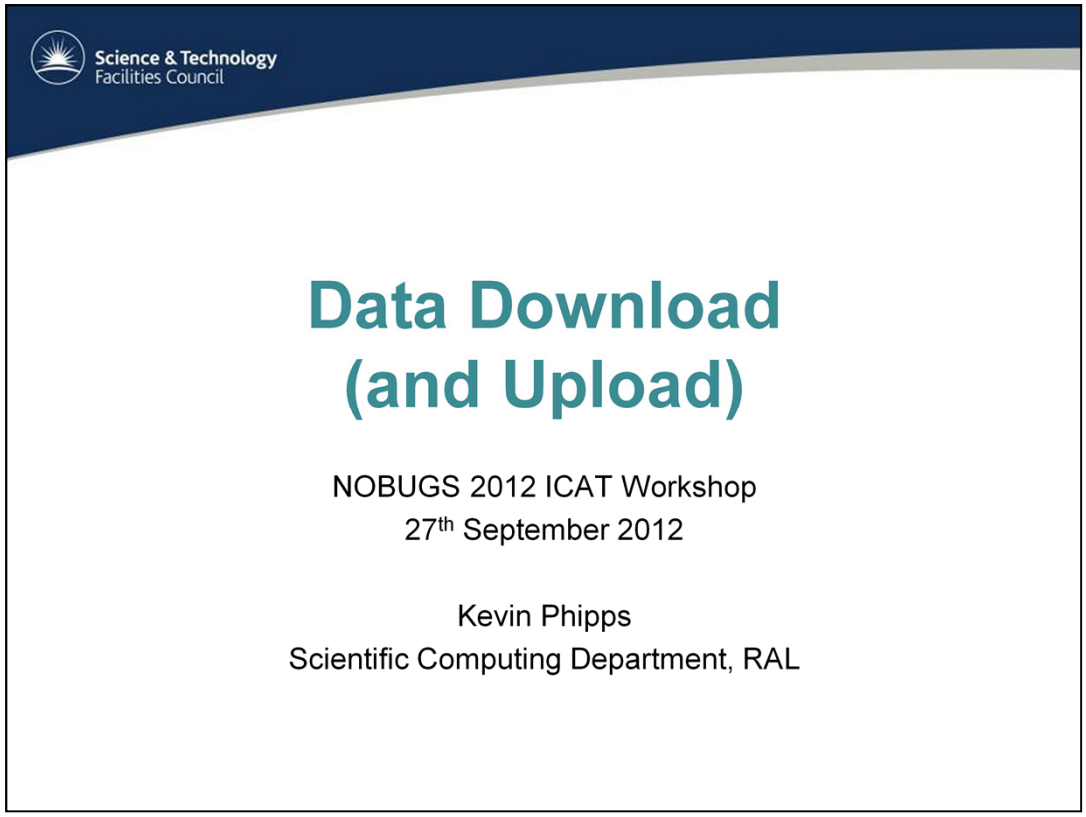
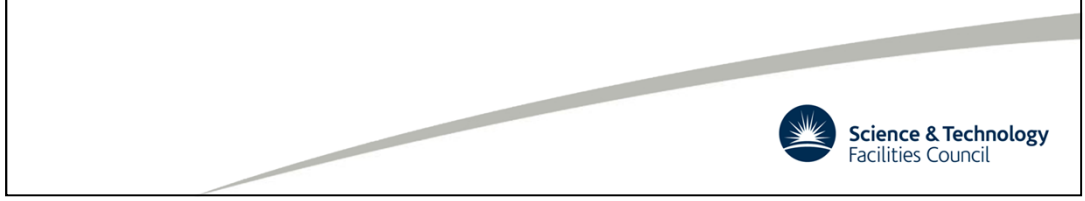


This talk is listed on the agenda as Data Download but it is also about Uploading data.

It is about how to handle the actual data files themselves.

Overview

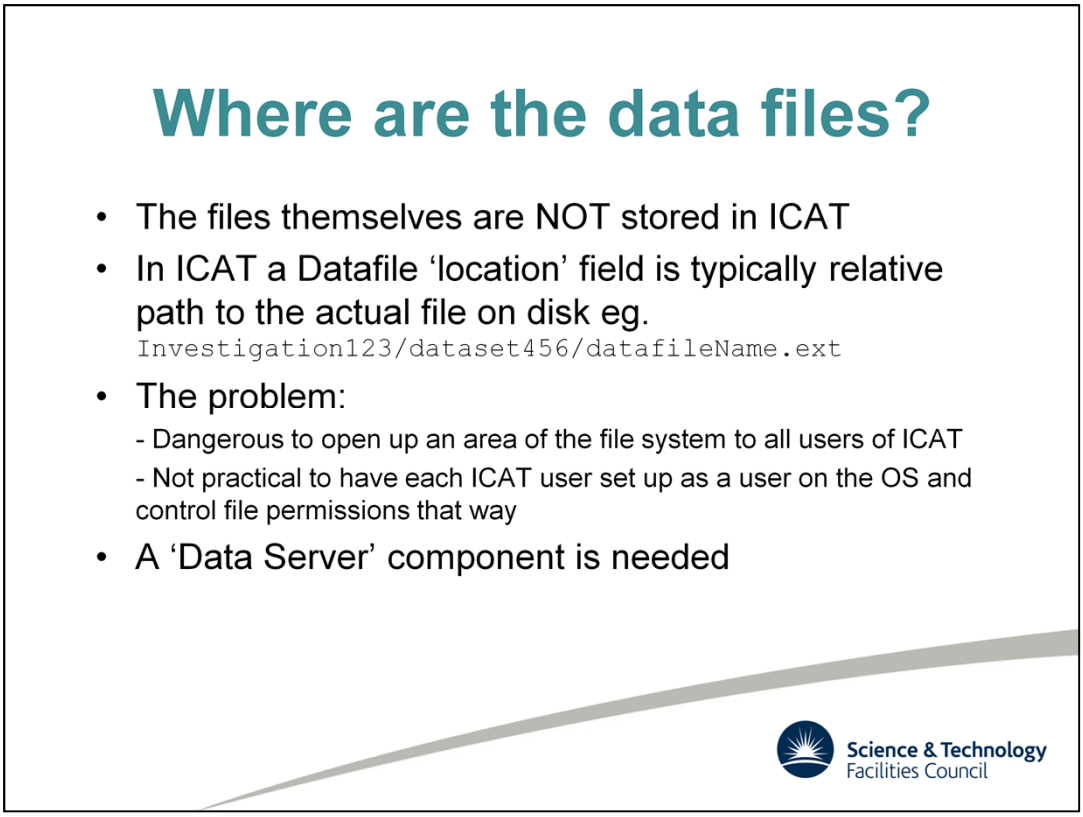
- Handling files in ICAT
- Why a data server is needed
- Proposal for a Data Server Interface Specification



Firstly, I will go over a few fundamental points about how to handle files in ICAT

Then, I will explain why a data server is needed to work with ICAT

Finally, I will outline a proposal that we have for a common Data Server Interface Specification



When you first start working with ICAT you are likely to think, so "Where are the data files?"

The important thing to realise is that the files themselves are not stored in ICAT.

To add a file to ICAT, you add a Datafile entry to a Dataset.

The Datafile object has a location field and here you would typically put the relative path to where the file will be located on disk, as shown in the example here.

The problem that you then have is how to control the access to the ICAT file store on the filesystem.

It is obviously dangerous to open up an area of the file system to all users of ICAT and

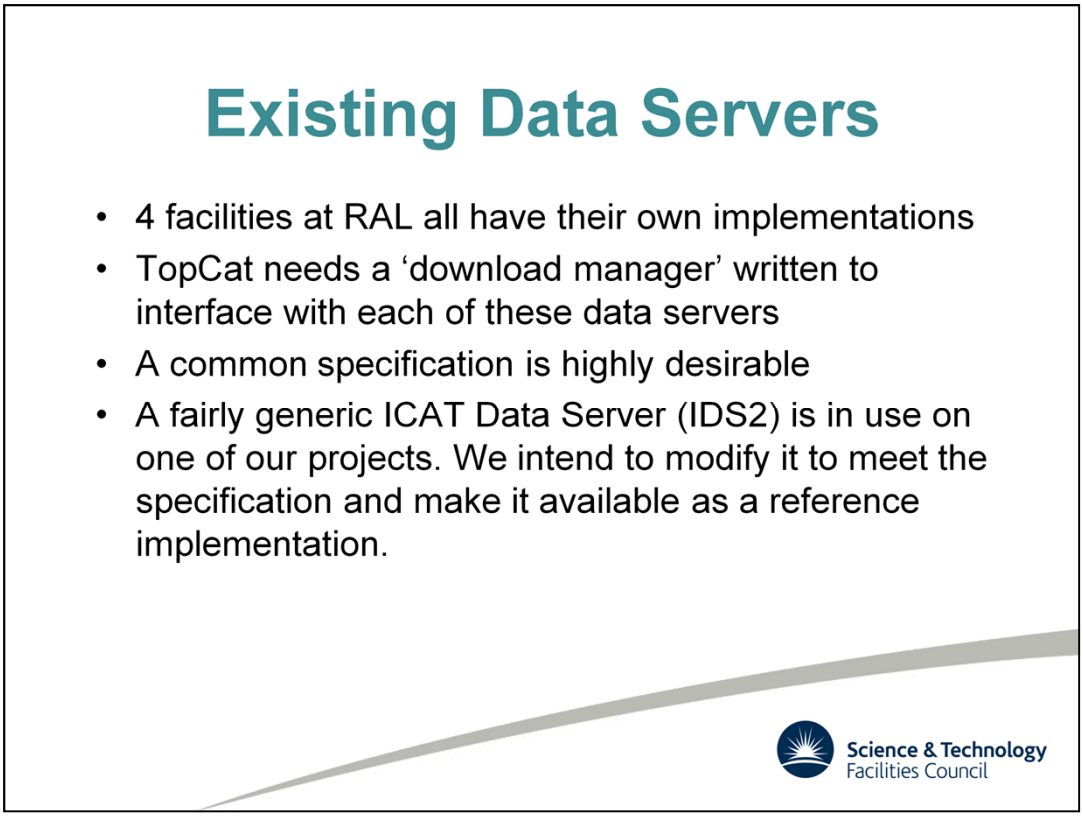
not practical to have each ICAT user set up as a user on the OS and control file permissions that way.

What is needed is a Data Server component that works alongside ICAT.

Data Server

- Currently a custom “Data Server” needs to be written to store, retrieve and delete the files and to communicate with ICAT for authorisation purposes
  - requests sent directly to data server
  - authorisation checked with ICAT
  - file stored, removed or returned
  - ICAT Datafile entry added or removed

Science & Technology Facilities Council



Recently, we took a look at what we are doing regarding data servers at RAL and found that 4 facilities here all have their own implementations.

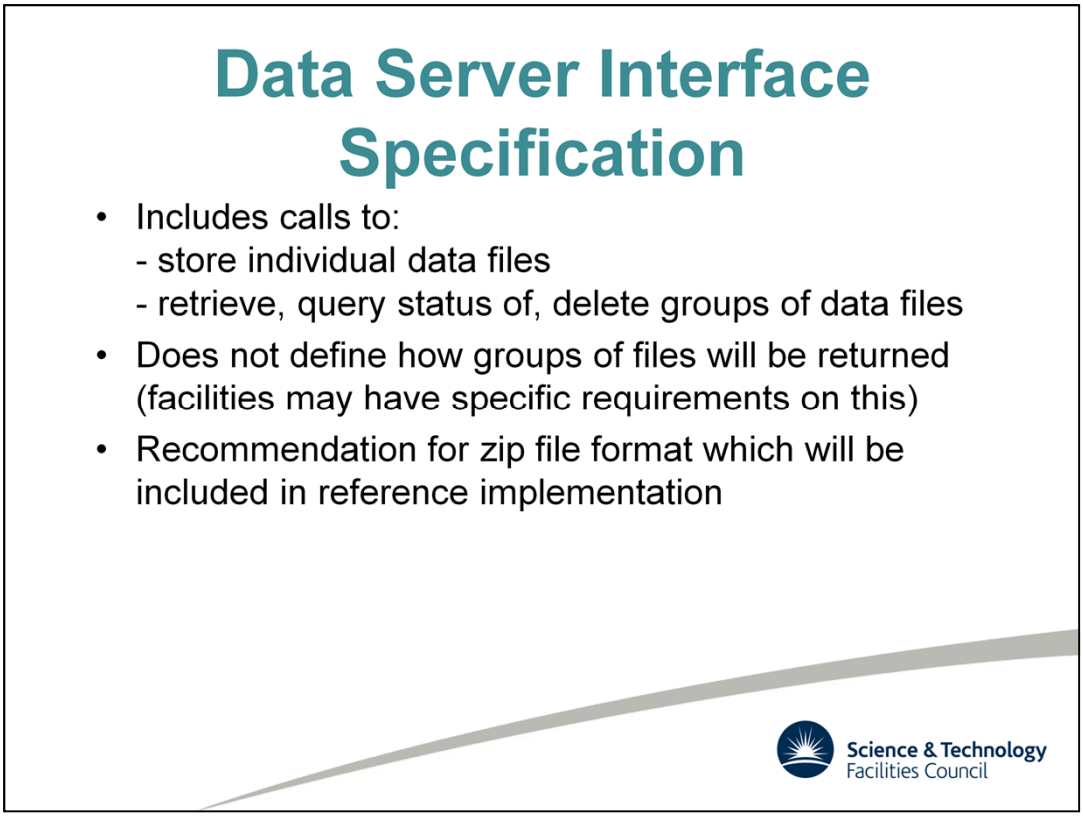
This is not so much of a problem until you look at tools like TopCat that work across multiple ICATs.

Currently, TopCat needs something called a 'download manager' written to interface with each of these data servers.

So you can see that a common specification is highly desirable.

What we do have is a fairly generic ICAT Data Server (IDS2) that is in use on one of our projects.

What we intend to do is modify it to meet a common specification and make it available as a reference implementation.



So, we have written down what we believe to be a workable Data Server Interface Specification which would at least cater for all the facilities here at RAL.

It includes calls to store individual data files, as well as to retrieve, query the status of and delete groups of data files.

What it does not define is how groups of files will be returned.

We found that even within the facilities we represented, each had specific requirements on

how the contents of a download would be structured and how the directories and files within it would be named.

Therefore, it seemed logical to leave this up to the implementation.

One thing which did seem to be worthwhile including was to have the option to request that downloads are zipped up

and this is something which we intend to include in the reference implementation.

ICAT Coupling

- ICAT session ID passed as argument to most calls
- For data retrieval requests the data server must check for **R**ead permissions in ICAT
- For put and delete requests it must check for **C**reate and **D**elete permissions and make corresponding changes in ICAT
- Consistency with ICAT must be maintained – orphan file preferred to an ICAT entry with no corresponding file

So how does the Data Server work alongside ICAT.

Well, one important thing is that an ICAT session ID needs to be passed as an argument to most calls.

For data retrieval requests the data server must check for **R**ead permissions in ICAT.

For put and delete requests it must check for **C**reate and **D**elete permissions as well as making the corresponding changes in ICAT.

The important thing is that consistency with ICAT must be maintained.

An orphan file preferred to an ICAT entry with no corresponding file on the data server which basically means

that the data server needs to make changes on the file system before registering these changes in ICAT.

Web Service Style

- Web service specification follows “REST” guidelines
- HTTP methods PUT, DELETE, POST and GET used
- POST supported in addition to GET for requests where URLs would become too long
- Calls taking parameters investigationIds, datasetIds and datafileIds use a comma separated list

Science & Technology Facilities Council

The specification we have put together is for a web service that follows the REST guidelines

and uses the HTTP methods PUT, DELETE, POST and GET .

POST is supported in addition to GET for requests where URLs would become too long.

Calls taking the parameters investigationIds, datasetIds and datafileIds use a comma separated list.

Dual Storage Model

- Based on assumption that there is a (fast) local cache of recently used data and a (slower) archive system
- Implementation needs to manage the cache of local files
- If one storage device is fast enough and large enough to hold all the data then this complexity is not required (but still follow the model)

Science & Technology Facilities Council

Something which the specification takes into account – and which is the case for 2 of the 4 facilities we support here –

is that the files may not always be immediately available from the underlying storage system.

So the specification is based on the assumption that there is a (fast) local cache of recently used data and a (slower) archive system.

What the data server implementation needs to do is manage the cache of local files – assuming that

the local cache is of a finite size and this size limit may be reached.

Of course, if you are in the fortunate position of having one storage device that is fast enough and

large enough to hold all the data then this complexity is not required.

But you should still follow the model to remain interoperable with other ICATs.

1) Upload a file to data server

- **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 contents of the file to be stored
  - Implementation also registers the file in ICAT
  - datafileModTime and datafileCreateTime must be in the format YYYY-MM-DD hh:mm:ss
  - **C**reate permission is required

Science & Technology Facilities Council

I will briefly go over the 6 methods that need to be implemented in order to create a compliant data server.

The first one deals with uploading a single file to the data server and is called put.

The various items of metadata for the file are set in the request header and

the body of the request is the contents of the file to be stored.

2) Delete file(s) from data server

- **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 dataserver and from ICAT
  - **D**elete permission is required

Science & Technology Facilities Council

To delete a file or files from the data server we have the delete method.

This utilises the HTTP DELETE method and a list of investigation Ids, dataset Ids and datafile Ids can be supplied.

3) Get / download file(s)

- **getData**

- http method: GET
- parameters: sessionIds, 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.

Science & Technology Facilities Council

The getData method is the one that actually returns a file bundle containing the requested files.

You can request that the download be compressed by setting the compress parameter.

4) Remove file from local storage

- **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.
  - For a single fast storage setup this method needs no implementation

Science & Technology Facilities Council

The remaining 3 methods only need implementing if your setup will be using a Dual Storage model.

To signal that you have finished working with a file and it can be removed from the local storage we have the archive method.

It is basically just for good housekeeping and helps to keep the size of the local cache down.

5) Make files available on local storage

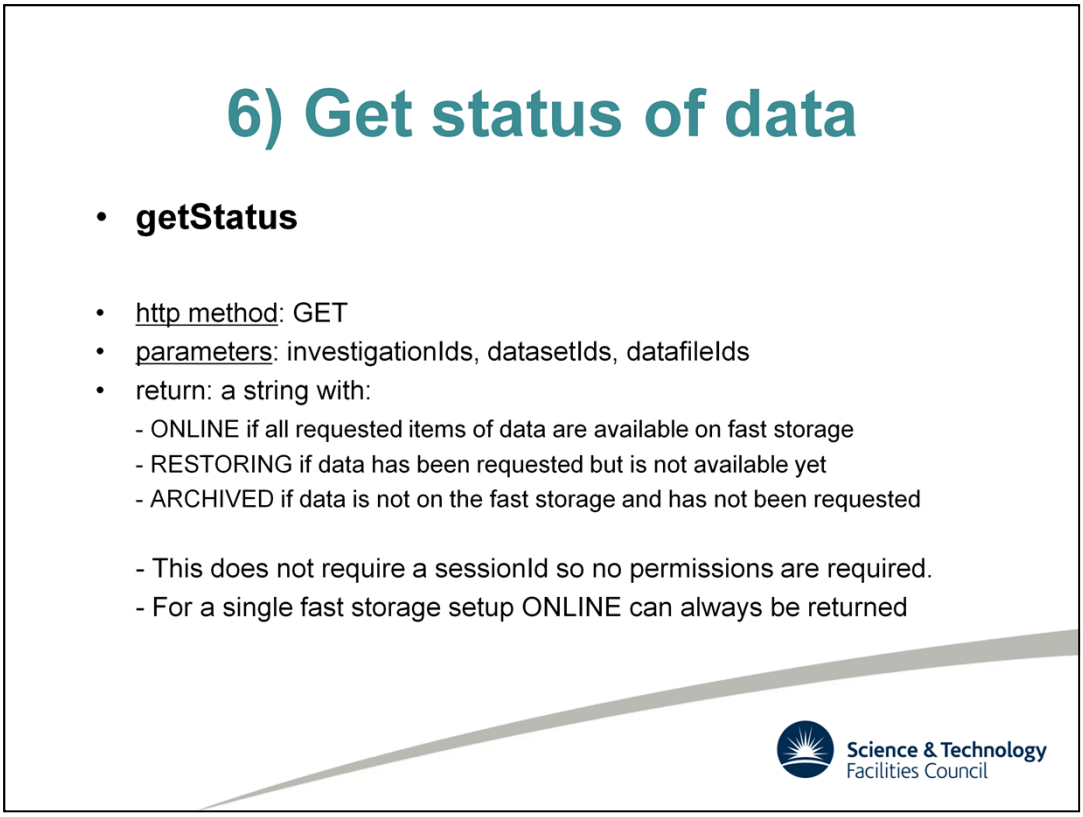
- **restore**

- http method: POST
  parameters: sessionIds, 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.
  - For a single fast storage setup this method needs no implementation

Science & Technology Facilities Council

In order to get files from the slower archive storage and make them available on the local storage we have the restore method.



In order to query the status of data you would like to download there is the getStatus method.

It returns a string with either:

ONLINE if all the requested items of data are available on the fast storage

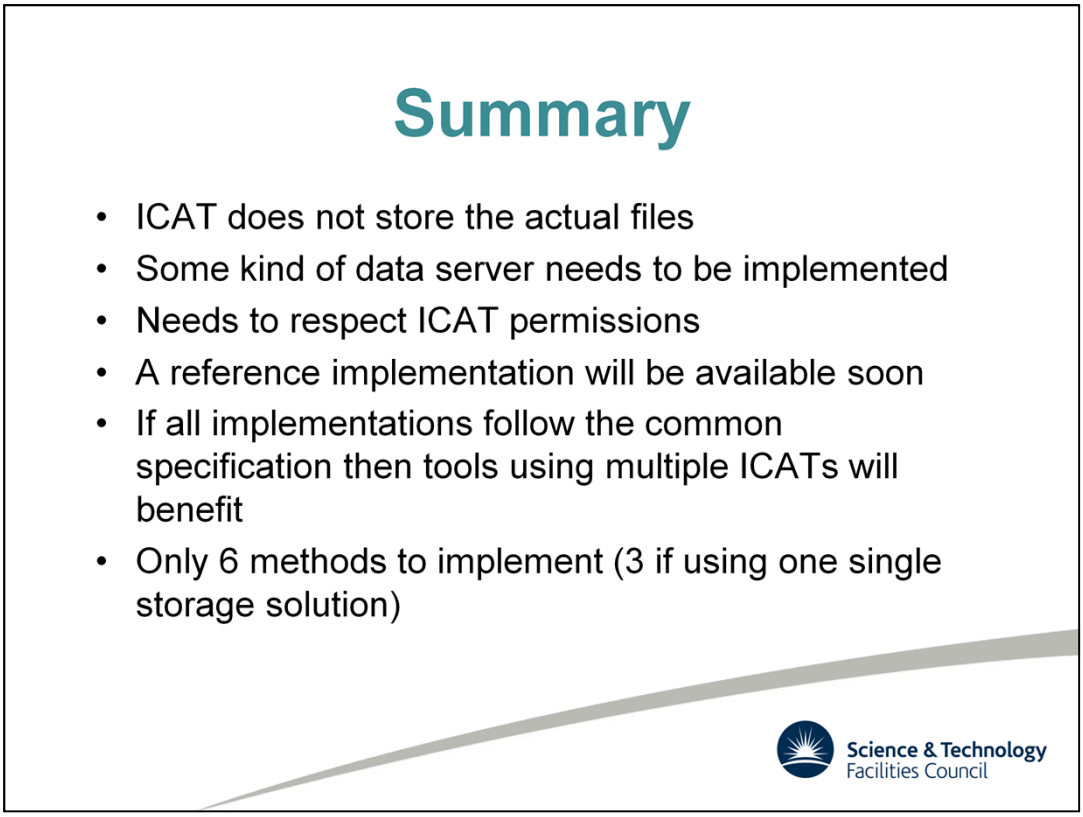
RESTORING if data has been requested but is not available yet on the fast storage or

ARCHIVED if the data is not on the fast storage and has not been requested

The typical usage scenario for these methods would be that you call the getData method to request some data for download.

If you get the response that not all of it is available then you can call the restore method to request that it be restored.

Then you need to keep polling the getStatus method until you get the response that it is ONLINE before requesting the download again.



So, in summary, it is important to realise that ICAT does not store the actual files themselves and that some kind of data server needs to be implemented.

This data server needs to respect ICAT permissions and always check with ICAT before performing any operation.

We are intending to make a reference implementation available soon so you can either use it, or use it as the basis for your own implementation.

If we can agree on a common specification and all implementations follow it then tools using multiple ICATs will all benefit.

Using the specification I have outlined there are only 6 methods to implement and only 3 if you are using one single storage solution.