How to Communicate with ICAT via SOAP

Chris Oliver,
Software Engineering Group,
Scientific Computing Department,
Rutherford Appleton Laboratory,
chris.oliver@stfc.ac.uk

March 2018

Abstract

This document describes and demonstrates how to communicate with an ICAT instance to find and download facility data. The general process for accessing data is described, which entails metadata retrieval from the ICAT Server, followed by data preparation and data retrieval from the ICAT Data Store (IDS). Then, specific implementations of this are shown. Firstly, communication using the ICAT Server SOAP interface together with the IDS REST API is demonstrated. Finally, the use of the Python-ICAT Python module to abstract away details of the SOAP interface is shown.

1 Introduction

The aim of this document is to describe and demonstrate how to communicate with an ICAT instance to find and download data.

Data can be stored in a facility in two ways; either on a disk in a cluster, or on a tape in an archive. This is referred to as two-level storage. Since the read time for tape-stored data is much longer than the read time for disk-stored data (minutes vs. seconds), ICAT is designed in a special way. For more information on ICAT, see the ICAT Project site [1].

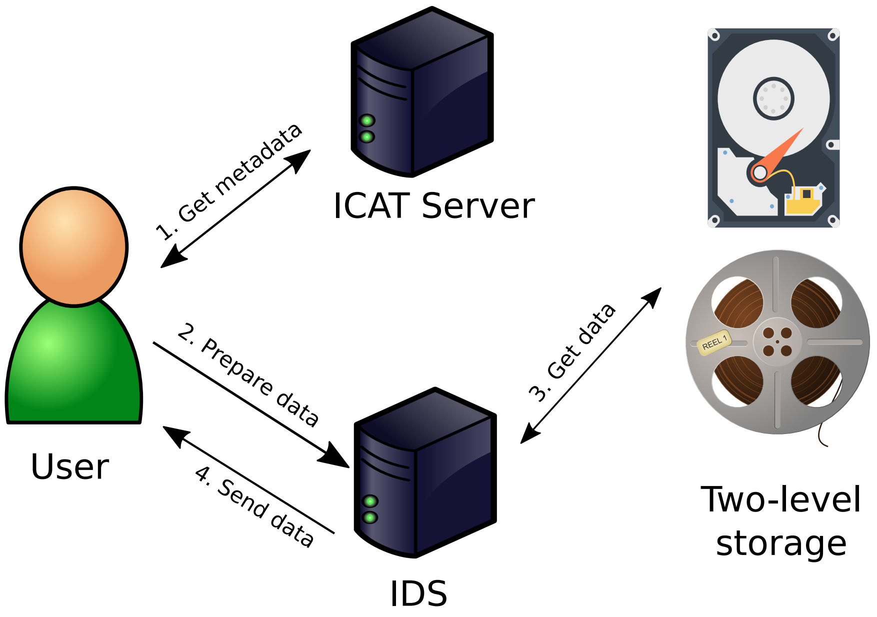

Figure 1: The process to find and download data from ICAT.

A typical ICAT instance has two main components: the ICAT Server and the ICAT Data Store (IDS). To download some data, the user first communicates with the ICAT Server to get metadata. They then communicate with the IDS, using the metadata to either retrieve the data almost instantly or to prepare it for download (i.e. read it from the tape), depending on how the data is stored. This is summarised in figure 1.

Communication with ICAT Server can be via REST or SOAP, and communication with the IDS is via REST only. A Python module, Python-ICAT, has been developed to act as a wrapper around the SOAP-based ICAT Server and REST-based IDS communications. It is useful if the application to be developed uses Python. An important point is that the ICAT Server REST API will soon be redesigned and the current version deprecated within about a year of writing. This means that any applications developed in that time-frame should perhaps be SOAP-based rather than REST-based. For this reason, the ICAT Server REST API is not discussed further in this document. For more information about these interfaces, see the online documentation [2, 3, 4, 5].

The data are represented in ICAT in a logical structure defined by the ICAT Schema. The full schema is complex, involving many entities and relationships, but for most purposes it is sufficient to know the following. The most basic entity is a datafile, which is a single file of data in some format. Multiple datafiles come together to form a dataset, and multiple datasets form an investigation.

An investigation corresponds to one experiment proposal (this is not part of the schema, but may help with understanding). Investigations have an attribute called a Visit ID, which a typical user would know. It is therefore a useful starting point in querying ICAT, as shall be shown later. For more information on the ICAT schema, see the online documentation [6].

The document is structured as follows. Section 2 shows how to communicate with ICAT using the ICAT Server SOAP interface, followed by the IDS REST API. Section 3 shows the same process but using the Python-ICAT wrapper for comparison, and is followed by a conclusion. The appendices show the Python code corresponding to Sections 2 and 3.

2 ICAT Server SOAP Interface

The Python code shown in Appendix A communicates with an ICAT instance to find all Visit IDs, and then find all associated datasets and datafiles before downloading selected datasets or datafiles.

The function `queryICAT` logs into ICAT using the user-provided username and password. It obtains a session ID and then uses this to make a number of queries, in JPQL syntax.

The first query obtains all visit IDs, and these are then iterated over to construct a Python dictionary of metadata about each visit. This dictionary is then returned.

The `fetchDataFromICAT` function uses this metadata to download user-specified datafiles. It passes the Session ID and dataset ID or datafile IDs to the IDS which then starts preparing the data for download. It returns a Prepared ID to represent the data to be prepared. The function then queries the IDS with the Prepared ID periodically to check if the data is ready yet. Once the data is prepared, it is downloaded via a get request. If the `filesList` is empty, the function downloads the whole dataset specified by `datasetID`. Otherwise, it downloads the specific files.

The function `copyfile` handles the actual copying of files, and is called by `fetchDataFromICAT`. The function `checkSessionId` will check if the session ID is still valid (they expire after around two hours). If not, it is renewed.

3 Python-ICAT

The code in Appendix B shows similar code to that in Appendix A, but using the Python-ICAT module to abstract away some details. It is clearly much shorter than the code in Appendix A, owing to the fact that Python-ICAT hides a lot of detail. However, the code is not able to download datasets properly. This is because, unlike the code in Appendix A, the `getPreparedData` function returns a Python file-like object. It is not clear how to handle these to produce zip files. For this reason, the code in Appendix B is not as well-developed as the code in Appendix A.

4 Conclusion

This document has described and demonstrated how to communicate with an ICAT instance to download data.

After a general introduction to how ICAT works, which involves querying the ICAT Server to obtain metadata, and then using this metadata to download data from the IDS, communication via the ICAT Server SOAP interface and the IDS REST API was described and demonstrated. Python code is supplied that retrieves all visit IDs and corresponding dataset and datafile metadata. It then downloads selected datasets and datafiles.

Then, the equivalent code but using the Python-ICAT Python module was shown. Some example Python code was provided that does a similar job to that of Appendix A, but is in a less well-developed state. This is caused by file downloads not working properly.

5 Appendix A: ICAT Server SOAP Interface Python Code

```
from suds.client import Client
import requests
import time
import zipfile
import StringIO

#Copy a file from infile to outfile. Used for saving single datafiles:
def copyfile(infile, outfile, chunksize=8192):

    while True:
        chunk = infile.read(chunksize)
        if not chunk:
            break
        outfile.write(chunk)

#Check if the time on sessionId is running out and refresh if it is:
def checkSessionId(sessionId):
   client = Client("https://icat02.diamond.ac.uk/ICATService/ICAT?wsdl")
   icat = client.service
   if (icat.getRemainingMinutes(sessionId) < 5):
      icat.refresh(sessionId)

#Queries ICAT Server to build a dictionary of visits, datasets and
    datafile metadata:
def queryICAT(fedID, userPassword):

   results = {"sessionID": "", "visits": []} #Initialise the dictionary
       to be returned

   client = Client("https://icat02.diamond.ac.uk/ICATService/ICAT?wsdl")
       #Get the WSDL file for SOAP. Currently uses the ICAT instance for
       public Diamond data
   icat = client.service
   factory = client.factory

   #Get a session ID:
   credentials = factory.create("login.credentials")
   entry = factory.create("login.credentials.entry")
   entry.key = "username"
   entry.value = fedID
   credentials.entry.append(entry)
   entry = factory.create("login.credentials.entry")
   entry.key = "password"
   entry.value = userPassword
   credentials.entry.append(entry)
   sessionId = icat.login("ldap", credentials)
   results["sessionID"] = sessionId

   visitIDs = icat.search(sessionId, "SELECT i.visitId FROM
       Investigation i") #Query for all visit IDs

   #Build the dictionary for each visit ID:
   visitDict = {"visitID": "", "visitName": "", "visitDate": "",
       "datasets": []}
   visitDictList = []
   for vid in visitIDs:

      visitDict["visitID"] = vid
      visitDict["visitName"] = icat.search(sessionId, "SELECT v.name
          FROM Investigation v where v.visitId=" + "'" + vid + "'")
      visitDict["visitDate"] = icat.search(sessionId, "SELECT
          v.startDate FROM Investigation v where v.visitId=" + "'" + vid
          + "'")

      datasetDict = {"ID": "", "path": "", "size": "", "date": "",
          "files": []}
      datasetDictList = []
      datasetIDs = icat.search(sessionId, "SELECT ds.id FROM Dataset ds
          WHERE ds.investigation.visitId=" + "'" + vid + "'")
      for did in datasetIDs:
         datasetDict["ID"] = did
         datasetDict["path"] = icat.search(sessionId, "SELECT ds.name
             FROM Dataset ds WHERE ds.id=" + "'" + str(did) + "'")
         datasetDict["date"] = icat.search(sessionId, "SELECT
             ds.startDate FROM Dataset ds WHERE ds.id=" + "'" + str(did)
             + "'")

         datafileDict = {"name": "", "size": "", "date": "", "ID": ""}
         datafileDictList = []
         datafileIDs = icat.search(sessionId, "SELECT df.id FROM
             Datafile df WHERE df.dataset.id=" + "'" + str(did) + "'")
         for dfid in datafileIDs:
            datafileDict["name"] = icat.search(sessionId, "SELECT df.name
                FROM Datafile df WHERE df.id=" + "'" + str(dfid) + "'")
            datafileDict["size"] = icat.search(sessionId, "SELECT
                df.fileSize FROM Datafile df WHERE df.id=" + "'" +
                str(dfid) + "'") #Size returned in bytes
            datafileDict["date"] = icat.search(sessionId, "SELECT
                df.createTime FROM Datafile df WHERE df.id=" + "'" +
                str(dfid) + "'")
            datafileDict["ID"] = dfid
            datafileDictList.append(datafileDict.copy())

         datasetDict["files"] = datafileDictList
         datasetDictList.append(datasetDict.copy())

      visitDict["datasets"] = datasetDictList
      visitDictList.append(visitDict.copy())

   results["visits"] = visitDictList
   checkSessionId(sessionId)

   return results

#Given the information in the dictionary, download datafiles and
    datasets using IDS:
def fetchDataFromICAT(sessionId, datasetID, filesList, outputDirectory):
    #filesList is a list of file IDs

   idsBaseURL = "https://ids01.diamond.ac.uk/ids"
   prepareDataURL = idsBaseURL + "/prepareData"
   isPreparedURL = idsBaseURL + "/isPrepared"
   getDataURL = idsBaseURL + "/getData"

   try:
      if (len(filesList) == 0):
```

```
    #Use the IDS REST API to download the data:

    #Tell IDS to prepare data for download:
    data = {"sessionId":str(sessionId),"datasetIds":datasetID}
    prepareDataResp = requests.post(prepareDataURL, data = data)
    preparedId = str(prepareDataResp.text)

    #Ping isPrepared every 30 seconds until data is ready:
    isPrepared ="false"
    isPreparedParams = {"preparedId":str(preparedId)}
    while (isPrepared == "false"):
       isPrepared = (requests.get(isPreparedURL,
           params=isPreparedParams)).text
       time.sleep(30)
       checkSessionId(sessionId)

    #Download the data to local directory:
    getDataParams = {"preparedId": preparedId}
    getDataResp = requests.get(getDataURL, params=getDataParams,
        stream=True)
    z = zipfile.ZipFile(StringIO.StringIO(getDataResp.content))
        #Store as zip file
    z.extractall(outputDirectory) #Extract zip file

 else:

    for f in filesList:

       #Use the IDS REST API to download the data:

       #Tell IDS to prepare data for download:
       data = {"sessionId":str(sessionId),"datafileIds": f}
       prepareDataResp = requests.post(prepareDataURL, data = data)
       preparedId = str(prepareDataResp.text)

       #Ping isPrepared every 30 seconds until data is ready:
       isPrepared ="false"
       isPreparedParams = {"preparedId":str(preparedId)}
       while (isPrepared == "false"):
          isPrepared = (requests.get(isPreparedURL,
              params=isPreparedParams)).text
          time.sleep(30)
          checkSessionId(sessionId)

       #Download the data to local directory:
       getDataParams = {"preparedId": preparedId}
       getDataResp = requests.get(getDataURL, params=getDataParams,
           stream=True)
       with open(outputDirectory + "/myData", 'wb') as f2:
```

```
                        copyfile(StringIO.StringIO(getDataResp.content),
                            f2)

      return "success"

   except:
      return "failed"
```

6 Appendix B: Python-ICAT Python Code

```
from icat.client import Client
import time
import zipfile
import os
import StringIO
import urllib2

#This function copies infile to outfile:

def copyfile(infile, outfile, chunksize=8192):

    while True:
        chunk = infile.read(chunksize)
        if not chunk:
            break
        outfile.write(chunk)

#Downloads a dataset given the ID:

def downloadDataset(datasetId):

   preparedId = myClient.prepareData({"Datasets":[datasetId]})
   print("preparedId:")
   print(preparedId)

   isPrepared = False
   while (isPrepared == False):
      print(isPrepared)
      isPrepared = myClient.isDataPrepared(preparedId)
      time.sleep(30)

   return myClient.getPreparedData(preparedId)

#Login:
```

```
myClient = Client("https://icat02.diamond.ac.uk/ICATService/ICAT?wsdl")
    #ICAT SOAP WSDL URL
credentials = {"username":"", "password":""} #Insert username and
    password
myClient.login("ldap", credentials)
myClient.add_ids("https://ids01.diamond.ac.uk/ids") #IDS URL

#Find all visit IDs:
visitIds = myClient.search("SELECT i.visitId FROM Investigation i")
print("Visit IDs found:")
print(visitIds)

#For each visit ID, list all datasets and metadata and download:
for vid in visitIds:

   #Get dataset IDs:
   datasetIdQuery = "SELECT ds.id FROM Dataset ds WHERE
       ds.investigation.visitId=" + "'" + vid + "'"
   datasetIds = myClient.search(datasetIdQuery)
   print("Dataset IDs found:")
   print(datasetIds)

   #Get dataset metadata
   print("Dataset metadata:")
   datasetMetaQuery = "SELECT ds FROM Dataset ds WHERE
       ds.investigation.visitId=" + "'" + vid + "'"
   datasetMetadata = myClient.search(datasetMetaQuery )
   print(datasetMetadata)

   print("Downloading datasets for this visit ID...")

   #Download all datasets for the visit ID:

   for i in range(0, len(datasetIds) - 1):

      getDataResp = downloadDataset(datasetIds[i])

      #I have tried several methods to save the dataset, none of which
          work. It produces gibberish rather than a zip file. The
          request returns a file-like object, which is different from
          what the requests library returns:
      '''
      z = zipfile.ZipFile(StringIO.StringIO(getDataResp.content)) #Store
          as zip file
      z.extractall() #Extract zip file

      with open("code2.zip", "wb") as code:

          code.write(myData.read())
```

```
with open("myData.out", 'wb') as f:

          copyfile(myData, f)
'''
print("Data downloaded to local directory")
```

References

[1] *The ICAT Project*[Online], Available at: https://icatproject.org/ [Accessed 14/03/2018].

[2] The ICAT Collaboration, *ICAT SOAP Manual*[Online], Available at: https://repo.icatproject.org/site/icat/server/4.9.1/soap.html [Accessed 14/03/2018], September 2017.

[3] *ICAT Restful API*[Online], Available at: https://repo.icatproject.org/site/icat/server/4.9.1/miredot/index.html [Accessed 14/03/2018], September 2017.

[4] *IDS Restful API*[Online], Available at: https://repo.icatproject.org/site/ids/server/1.8.0/miredot/index.html [Accessed 14/03/2018], August 2017.

[5] *PYTHON ICAT*[Online], Available at: https://icatproject.org/user-documentation/python-icat/ [Accessed 14/03/2018].

[6] *ICAT SCHEMA*[Online], Available at: https://icatproject.org/user-documentation/icat-schema/ [Accessed 14/03/2018].