Common API Responses

Route: /status/:external_uri
Header: {service: , events: :all, :none (defaults to :none)}
Example: /status/glacier_id, header: {service: glacier}
Responses:
Code: 200
{external_uri:
staged: boolean
staged_location: (if exists)
service:
events: [event1, event2, event3, etc]
}

Event format: {type: pending/running/succeeded/failed, action: stage/cancel/etc, vendor_message: (any info from vendor, including vendor errors), timestamp:}

Event log is nil if no commands have ever been run, or at least do not exist in the database

Route: /stage/:external_uri
Header: {service: }
Example: /stage/glacier_id header: {service: glacier}
Responses:
Code: 200
{
external_uri:
service:
event: the singular stage event
}

Camel

https://github.com/samvera-labs/external-storage-proxy

Jetty

- …

JDBC

Resources

http://camel.apache.org/sql-example.html
http://camel.apache.org/download.html

Object 1: Establish Camel Route to update/read from database

1. Install local MySQL or PostgreSQL
2. Create databases and tables
3. Demonstrate INSERT and SELECT

Table: Jobs
Fields:

- External URI: varchar
- Fedora URI: varchar
- Staged: int
- Staged Location: varchar
- Service: varchar

Table: Events
Fields:

- External URI (pk): varchar
- Type: varchar (example: “queued”, “pending”)
- Action: varchar (request verb)
- Vendor_Message: varchar
- Timestamp: datetime (of event)

DB Setup

```
create database external_storage_proxy;
use external_storage_proxy;

create table jobs (
ID int NOT NULL AUTO_INCREMENT PRIMARY KEY,
EXTERNAL_URI VARCHAR(255) NOT NULL,
FEDORA_URI VARCHAR(255) NOT NULL,
```

```
STAGED int(2) UNSIGNED ,
STAGED_LOCATION varchar(255),
SERVICE varchar(47)
);
```

```
create table events (
ID int NOT NULL AUTO_INCREMENT PRIMARY KEY,
TYPE varchar(47) NOT NULL,
ACTION varchar(47) NOT NULL,
VENDOR_MESSAGE text,
EVENT_TIME TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
JOB integer REFERENCES jobs
);
```

```
INSERT INTO jobs (EXTERNAL_URI, FEDORA_URI, SERVICE, STAGED)
VALUES (“1234uuidabc”, “123ab45cd”, “s3”, 0);

INSERT INTO events (TYPE, ACTION, JOB)
VALUES (“pending”, “stage”, 1);
```

Insert into DB via Camel:

Karaf console

```
feature:repo-add camel 2.19.2
feature:install camel
feature:install camel-sql
install -s mvn:commons-pool/commons-pool/1.6
install -s
mvn:org.apache.servicemix.bundles/org.apache.servicemix.bundles.commons-dbcp/1.4_3
feature:install pax-jdbc-mysql
feature:install camel-jetty
```

Route deploy

Copy db.xml to karaf deploy directory
Edit db.xml to add database username and password.
http://localhost:9091/say/hello

http://localhost:9091/say/bye

Don's Notes on API Revisions

Use Cases

- Program can

FileStatus

old Route: GET /status/:external_uri
proposed route: GET /:service/status?external_uri=
-or- /:service/status/*external_uri
Header: {events: :all, :none (defaults to :none)}
Example: /glacier/status/glacier_id, header: {}
Responses:

Code: 200
{external_uri:
staged: 0 or 1 (0 = not staged, 1 = staged)
staged_location: (optional. only present if exists)
service:
events: [event1, event2, event3, etc]
}

Event format: {type: pending/running/succeeded/failed, action: stage/cancel/etc, vendor_message: (any info from vendor, including vendor errors, formatted as a string), timestamp:}

Event log is empty (i.e. [ ]) if there are is no event log in the database

StageFile

old Route: POST /stage/:external_uri
proposed route: POST /:service/stage/*external_uri
Header: { }
Example: /glacier/stage/glacier_id header: {}
Responses:

Code: 200
{

external_uri:
service:
event: the singular stage event
}

Wed. Afternoon Proposal:

(Chris Colvard, Carrick Rogers, and Randall Floyd present)

Route: /status/:external_uri
Header: {service: , actions: :all, :stage, :fixity (defaults to [:all])}
Response:

```
Code: 200
{
 external_uri:
 service:
 <action_name>: {
                  status: pending | success | failure
                  result:
                  vendor_message:
                  created_at:
                  updated_at:
                 }
}
```

Status contains the external_uri and service that it was called with along with information from the last known status of actions that have been run on the file. Each action contains a status, a result (if status is not pending), a vendor_message (if supplied by the vendor), a create time, and an update time. A status request of a file that has never had an action performed on it would return a 200 with only an external_uri and service.

Route: /stage/:external_uri
Header: {service: }
Response:

```
Code: 200
{
external_uri:
service:
stage: {
        status: pending | success | failure
        result:
        vendor_message:
        created_at:
        updated_at:
       }
}
```

The difference between the results of status and stage is that the stage result will only hold a single action hash: that of stage.

Examples:
Route: /status/http%3A%2F%2Fs3.amazonaws.com%2Fbucket%2Ffile
Header: {service: ‘s3’}
Response:

Code: 200

```
{
 “external_uri”: “http://s3.amazonaws.com/bucket/file”
 “service”: “s3”
 “stage”: {
            “status”: “success”
            “result”: “http://s3.amazonaws.com/staging-bucket/file”
            “vendor_message”: “Performed at a cost of $0.01”
            “created_at”: “2017-09-11 09:42:34 -0400”
            “updated_at”: “2017-09-12 09:42:34 -0400”
           }
}
```

Route: /status/http%3A%2F%2Fs3.amazonaws.com%2Fbucket%2Ffile
Header: {service: ‘s3’}
Response:

Code: 200

```
{
 “external_uri”: “http://s3.amazonaws.com/bucket/file”
 “service”: “s3”
 “stage”: {
            “status”: “pending”
            “created_at”: “2017-09-12 09:42:34 -0400”
            “updated_at”: “2017-09-12 09:42:34 -0400”
           }
 “fixity”: {
            “status”: “success”
            “result”: “d41d8cd98f00b204e9800998ecf8427e”
            “vendor_message”: “Performed at a cost of $0.02”
            “created_at”: “2017-09-11 08:42:34 -0400”
            “updated_at”: “2017-09-11 09:42:34 -0400”
           }
}
```

This example shows how each action is independent as each vendor may or may not require that the file be staged in order to run fixity. In the example above a fixity was run and then a stage requested. If a user requests /fixity at this point then the fixity status would flip to “pending”, the timestamps would be updated, and the result and vendor_message would be cleared.

Route: /stage/http%3A%2F%2Fs3.amazonaws.com%2Fbucket%2Ffile
Header: {service: ‘s3’}
Response:

Code: 200

```
{
 “external_uri”: “http://s3.amazonaws.com/bucket/file”
 “service”: “s3”
 “stage”: {
            “status”: “pending”
            “created_at”: “2017-09-12 09:42:34 -0400”
            “updated_at”: “2017-09-12 09:42:34 -0400”
           }
}
```

Implementation Considerations:

It is the responsibility of the implementation to ensure that the details of each action returned in a status result to not be misleading. If a file has been staged and the staged copy deleted then the status should not return a stage action with a status of success and a result. A possible approach to aide in expiring action results would be to make each action “stage” a file. For example, a fixity action would stage a fixity result file (e.g. file.md5). Using this it is easy to tell if an action should be expired by simply checking for the existence of this file. (This could be a local file system check for existence or a read of the first few bytes of an http resource.) This test is a good sanity check but can be expensive so it could be toggled via a configuration option. An implementation or vendor may be able to provide you this information via a file system watcher or a callback so it would not have to happen during the request cycle.