inBloom

Learning Map and inBloom Index System

Data Model and Server

inBloom Index **Administrator Guide**

22 Jan 2013

Applied Minds, LLC

1209 Grand Central Ave Glendale CA 91201

Contents

Background

The Learning Map Data Model

The Bill & Melinda Gates Foundation (the Foundation) supports the implementation of the Common Core State Standards for US K-12 education. The Foundation awarded a contract to Applied Minds, LLC (AMI) to develop a Learning Map Data Model (LMDM) with the goal of it becoming a standard for educational technology infrastructure. The Learning Map will provide the organizing framework that maps the relationship between learning objectives, including dependencies and higher level groupings. It will also allow educational media resources, such as courses, books and web content to be linked to learning objectives. Curricula aligned to the standards will exhibit great diversity in highlighting paths through the Learning Map, and a suite of tools will allow authoring and visualization of the Learning Map. We believe that this data model will eventually enable the creation of online learning tools that are more responsive to the individual needs of a student. The LMDM is inspired by and based on the philosophy of the more general Knowledge Web (See: http://edge.org/conversation/aristotle-the-knowledge-web).

inBloom Technology

The Foundation has, in collaboration with the Carnegie Corporation, initiated an ambitious effort, Shared Learning Infrastructure (SLI), now inBloom Technology, to provide a new technology infrastructure that supports the Common Core Standards and the Foundation's vision, to be implemented by inBloom. AMI has been contracted by inBloom to build an implementation of a Learning Map and Learning Registry System, suitable for third-party software developers to populate content and develop applications.

InBloom Index Administrator Guide

As part of this effort, AMI has delivered an inBloom Index Server, formerly called the Learning Registry Index (LRI) Server. This document provides useful information for administrators of this server, and includes

1. Installation Guide
2. Command Line Administration Guide
3. Configuration Guide
4. Enumeration Guide
5. Query Sequencer Guide
6. Schema Update Guide

inBloom Index Installation Guide

Dependencies

Environment

1. You need to allocate a directory to hold the inBloom Index server code. We'll call it the `$LRIDIR` variable.

neo4j

1. Grab neo4j from `http://neo4j.org/` (community edition) and untar it into `$DBDIR`.
2. Edit the neo4j server config to set port numbers and server address (if needed). Example:

```
vi LRIDIR/neo4j-community-1.8/conf/neo4j-server.properties

org.neo4j.server.webserver.address=192.168.100.75
org.neo4j.server.webserver.port=7000
org.neo4j.server.webserver.https.port=7001
```

uwsgi

Make sure you've got it installed. We'll be using a local config, so your system-wide config can be safely ignored. Example local config in `$LRIDIR/wsgi_config.json` to work with nginx:

```
{
  "uwsgi": {
      "gid":"www-data",
      "uid":"www-data",
      "enable-logging":true,
      "socket":"/tmp/lri_sandbox.wsgi.sock",
      "chmod-socket":666,
      "protocol":"wsgi",
      "buffer-size":25000,
      "wsgi-file":"/home/lri/sandbox/webapp.py",
      "pyargv":"/home/lri/sandbox/lri_config.json",
      "plugins":["python","http"],
      "processes":1,
      "master":true,
      "harakiri":60,
      "limit-as":1024,
      "memory-report":true,
      "no-orphans":true
  }
}
```

or run standalone:

```
{
  "uwsgi": {
      "gid":"www-data",
      "uid":"www-data",
      "enable-logging":true,
      "socket":"192.168.100.75:8000",
      "chmod-socket":666,
      "protocol":"http",
      "buffer-size":25000,
      "wsgi-file":"/home/lri/sandbox/webapp.py",
      "pyargv":"/home/lri/sandbox/lri_config.json",
      "plugins":["python","http"],
      "processes":1,
      "master":true,
      "harakiri":60,
      "limit-as":1024,
      "memory-report":true,
      "no-orphans":true
  }
}
```

nginx (optional)

If you want a real webserver that will integrate with uwsgi, nginx is recommended. Here's a sample `sites-available` config for nginx that you may find useful.

```
server {
    listen   8000; ## listen for ipv4; this line is default and implied

    server_name 192.168.100.75;

    location / {
                include uwsgi_params;
                uwsgi_pass unix:/tmp/lri_sandbox.wsgi.sock;
                uwsgi_param UWSGI_CHDIR $LRIDIR;
                uwsgi_param UWSGI_SCRIPT webapp;
    }

}
```

memcached

You could start it with something like:

```
/usr/bin/memcached -d -m 256 -p 11211 -u memcache -l 192.168.100.75
```

if your system config is non-existant.

NOTE: You must also install the `python-memcache` module.

inBloom Index Setup Steps

1. Unpack the inBloom Index repo into $LRIDIR.
2. Edit the `lri_config.json` file. Example:

```
{
 "neo4j_dir":"/home/lri/sandbox/neo4j-community-1.8.1",
 "neo4j_server_host":"192.168.100.75",
 "neo4j_server_port":7000,
 "slc_server_host":"api.sandbox.slcedu.org",
 "slc_token_check_path":"/api/rest/system/session/check",
 "cache_type":"memcached",
 "memcached_host":"192.168.100.75",
 "memcached_port":"11211",
 "bootstrap_filenames":["/home/kurt/neo/lri_schema/bootstrap.json",
                        "/home/kurt/neo/lri_schema/cc_schema.json"],
 "default_creator_guid":"LRI_TEST_CREATOR",
 "admin_passwd":"fr00tlupe",
 "use_cache":true,
 "admin_access_tokens":{"letmein":"LRI_ADMIN_USER_0"}
}
```

1. Make sure your `JAVA_HOME` is set in your `~/.bashrc`. Example:
    - `JAVA_HOME=/home/lri/jdk1.6.0_30`
2. Edit the neo4j, uwsgi, nginx, and memcached configs as needed.
3. Start nginx and memcached if using them.
4. Initalize the neo4j server:
    - `$LRIDIR/lri_admin.py init_neo4j`
5. Start the neo4j server:
    - `$LRIDIR/lri_admin.py start_neo4j`
6. Initalize the LRI server by loading the bootstrap schemata:
    - `$LRIDIR/lri_admin.py create_lri`
7. Start the LRI server:
    - `$LRIDIR/lri_admin.py start_lri`
8. Switch to another terminal/screen.
9. Load the CCSS entities:
    - `$LRIDIR/query_sequencer.py 192.168.100.75:8000 $LRIDIR/test_suites/load/ccss_suite_new.yaml`
10. Enjoy!

Command Line Administration Guide

lri_admin.py

lri_admin.py is a command line tool for administering an instance of the inBloom Index. It is used to start the inBloom Index, start and stop neo4j, and manage database snapshots for backup purposes.

Invocation syntax

```
./lri_admin.py <COMMAND> <ARGUMENT (for some commands)>
```

This tool has the following commands:

init_neo4j

When setting up an inBloom Index instance for the first time, the neo4j distribution should be untarred into the directory specified by the neo4j_dir parameter in the lri_config.json file. Once neo4j is properly installed, the init_neo4j command should be run to to configure its use with the inBloom Index server.

Example:

```
./lri_admin.py init_neo4j
```

Response:

```
SHELL: mv "/home/kurt/neo/neo4j-community-1.8/data"
"/home/kurt/neo/neo4j-community-1.8/data.LIVE"
SHELL OUTPUT:  0
SHELL: ln -s "/home/kurt/neo/neo4j-community-1.8/data.LIVE"
"/home/kurt/neo/neo4j-community-1.8/data"
SHELL OUTPUT:  0
SHELL: /home/kurt/neo/neo4j-community-1.8/bin/neo4j stop
SHELL OUTPUT: ERROR: Neo4j Server not running 0
SHELL: cp -pr /home/kurt/neo/neo4j-community-1.8/data.LIVE
/home/kurt/neo/neo4j-community-1.8/data.snapshot.ORIG
SHELL OUTPUT:  0
[
    "SNAPSHOT TAKEN --> ORIG"
]
NUMBER OF ERRORS FOUND = 0
```

start_neo4j

This starts the neo4j server process, but not the inBloom Index server.

Example:

```
./lri_admin.py start_neo4j
```

Response:

```
SHELL: /home/kurt/neo/neo4j-community-1.8/bin/neo4j start
```

```
SHELL OUTPUT: WARNING: Max 4096 open files allowed, minimum of 40 000
recommended. See the Neo4j manual.
Starting Neo4j Server...WARNING: not changing user
process [27208]... waiting for server to be ready....... OK.
Go to http://localhost:7474/webadmin/ for administration interface. 0
[]
NUMBER OF ERRORS FOUND = 0
```

stop_neo4j

This stops the neo4j server process, but not the inBloom Index server.

Example:

```
./lri_admin.py stop_neo4j
```

Response:

```
SHELL: /home/kurt/neo/neo4j-community-1.8/bin/neo4j stop
SHELL OUTPUT: Stopping Neo4j Server [27208]..... done 0
[]
NUMBER OF ERRORS FOUND = 0
```

create_lri

This command initializes a new inBloom Index server in a fresh, empty neo4j database. Initialization consists of loading of the bootstrap schema as declared in the `bootstrap_filenames` parameter of `lri_config.json`. `start_neo4j` should be run before this command, if neo4j is not already running. The command usually takes a long time to run. The inBloom Index debugging log file `LRI_DEBUG.log` in the `neo4j_dir` directory shows much detail as the command proceeds.

Example:

```
./lri_admin.py create_lri
```

Response:

```
NUMBER OF ENTITIES = 133
ERRORS FROM INITIALIZATION BOOTSTRAP:
WARNINGS FROM INITIALIZATION BOOTSTRAP:
SCHEMA INDEX STATS:
datatypes has size = 26
guid has size = 0
properties has size = 118
types has size = 40
id has size = 133
NUMBER OF ERRORS FOUND = 0
```

start_lri

This starts the inBloom Index server.

Example:

```
./lri_admin.py start_lri
```

Response:

```
RUNNING WSGI SERVER WITH: uwsgi --json "./wsgi_config.json"
tmp = /
[uWSGI] getting JSON configuration from ./wsgi_config.json
/usr/lib/uwsgi/plugins/python27_plugin.so
*** Starting uWSGI 0.9.8.1-debian (64bit) on [Thu Jan 17 15:59:35 2013]
***
compiled with version: 4.6.1 on 28 June 2011 10:48:13
limiting address space of processes...
your process address space limit is 1073741824 bytes (1024 MB)
your memory page size is 4096 bytes
 *** WARNING: you have enabled harakiri without post buffering. Slow
upload could be rejected on post-unbuffered webservers ***
uwsgi socket 0 bound to UNIX address /tmp/wsgi.sock fd 3
Python version: 2.7.2+ (default, Oct  4 2011, 20:41:12)  [GCC 4.6.1]
Python main interpreter initialized at 0x1cc3590
your server socket listen backlog is limited to 100 connections
*** Operational MODE: single process ***
WSGI application 0 (SCRIPT_NAME=) ready on interpreter 0x1cc3590 pid:
29945 (default app)
*** uWSGI is running in multiple interpreter mode ***
spawned uWSGI master process (pid: 29945)
spawned uWSGI worker 1 (pid: 29949, cores: 1)
```

take_snapshot

This stops the inBloom Index and neo4j servers, make a copy of the database, and then restarts both servers. Snapshots serve as archivals backups, and are used as part of the inBloom Index schema update mechanism.

Example:

```
./lri_admin.py take_snapshot
```

takes a snapshot and names it with the current time.

Response:

```
SHELL: /home/kurt/neo/neo4j-community-1.8/bin/neo4j stop
SHELL OUTPUT: Stopping Neo4j Server [27511]..... done 0
SHELL: cp -pr /home/kurt/neo/neo4j-community-1.8/data.LIVE
/home/kurt/neo/neo4j-community-1.8/data.snapshot.2013-01-17T23:40:08
SHELL OUTPUT:  0
SHELL: /home/kurt/neo/neo4j-community-1.8/bin/neo4j start
SHELL OUTPUT: WARNING: Max 4096 open files allowed, minimum of 40 000
recommended. See the Neo4j manual.
Starting Neo4j Server...WARNING: not changing user
process [28342]... waiting for server to be ready.... OK.
Go to http://localhost:7474/webadmin/ for administration interface. 0
```

```
[
    "SNAPSHOT TAKEN --> 2013-01-17T23:40:08"
]
NUMBER OF ERRORS FOUND = 0
```

The snapshot is named `2013-01-17T23:40:08` in this example.

restore_snapshot

This stops the inBloom Index and neo4j servers, and restores the named snapshot to being live, and then restarts both servers. The previously live version of the database is marked as deleted at the current time.

Example:

```
./lri_admin.py restore_snapshot 2013-01-17T23:40:08
```

takes a snapshot and names it with the current time.

Response:

```
SHELL: /home/kurt/neo/neo4j-community-1.8/bin/neo4j stop
SHELL OUTPUT: Stopping Neo4j Server [29374]..... done 0
SHELL: mv /home/kurt/neo/neo4j-community-1.8/data.LIVE
/home/kurt/neo/neo4j-community-1.8/DELETED_AT.2013-01-17T23:54:31
SHELL OUTPUT:  0
SHELL: cp -pr /home/kurt/neo/neo4j-community-1.8/data.snapshot.2013-01-
17T23:40:08 /home/kurt/neo/neo4j-community-1.8/data.LIVE
SHELL OUTPUT:  0
SHELL: /home/kurt/neo/neo4j-community-1.8/bin/neo4j start
SHELL OUTPUT: WARNING: Max 4096 open files allowed, minimum of 40 000
recommended. See the Neo4j manual.
Starting Neo4j Server...WARNING: not changing user
process [29613]... waiting for server to be ready.... OK.
Go to http://localhost:7474/webadmin/ for administration interface. 0
[
    "LIVE deleted ---> DELETED_AT.2013-01-17T23:54:31",
    "SNAPSHOT RESTORED <--- 2013-01-17T23:40:08"
]
NUMBER OF ERRORS FOUND = 0
In this example, the previously live database is renamed to `DELETED_AT.2013-01-17T23:54:31`. ## `reset` This restores a snapshot of an empty neo4j database.
```

Example:

```
./lri_admin.py reset
```

Response:

```
SHELL: /home/kurt/neo/neo4j-community-1.8/bin/neo4j stop
SHELL OUTPUT: Stopping Neo4j Server [29613]..... done 0
SHELL: mv /home/kurt/neo/neo4j-community-1.8/data.LIVE
/home/kurt/neo/neo4j-community-1.8/DELETED_AT.2013-01-17T23:57:05
SHELL OUTPUT:  0
```

```
SHELL: cp -pr /home/kurt/neo/neo4j-community-1.8/data.snapshot.ORIG
/home/kurt/neo/neo4j-community-1.8/data.LIVE
SHELL OUTPUT:  0
SHELL: /home/kurt/neo/neo4j-community-1.8/bin/neo4j start
SHELL OUTPUT: WARNING: Max 4096 open files allowed, minimum of 40 000
recommended. See the Neo4j manual.
Starting Neo4j Server...WARNING: not changing user
process [29857]... waiting for server to be ready....... OK.
Go to http://localhost:7474/webadmin/ for administration interface. 0
[
    "LIVE deleted ---> DELETED_AT.2013-01-17T23:57:05",
    "SNAPSHOT RESTORED <--- ORIG"
]
NUMBER OF ERRORS FOUND = 0
```

validate_live_schema

This runs a suite of validation tests on the schema in the live database.

Example:

```
./lri_admin.py validate_live_schema
```

Response:

```
NUMBER OF ERRORS FOUND = 0
```

validate_bootstrap_schema

This runs a suite of validation tests on the schema in schema files listed by the `bootstrap_schema` parameter of the `lri_config.json` configuration file.

Example:

```
./lri_admin.py validate_bootstrap_schema
```

Response:

```
NUMBER OF ERRORS FOUND = 0
```

regression

This runs a suite of regression tests in the `test_suites/regression` directory

Example:

```
./lri_admin.py regression
```

Response:

```
Running Regression Tests....
7 test sequences to run.
START test sequence: ./test_suites/regression/suite_01.......
Running Step: step000
```

Configuration Guide

The inBloom Index is configured by the use of a central configuration file, `lri_config.json`, and a UWSGI config file `usgi_config.json`.

`lri_config.json`

This is the central configuration file for the inBloom Index server. Configuration is determined by the value of the configuration parameters. In general, there are no default values. Missing parameters may cause sub-optimal operation.

`host`

This string value is the host IP address/name for the inBloom Index server itself.

`port`

This integer value is the host port for the inBloom Index server itself.

`neo4j_dir`

This string value is the full pathname of the neo4j installation (where it was untarred).

`neo4j_server_host`

This string value is the host IP address/name for the neo4j server. This value should match the `org.neo4j.server.webserver.address` parameter value in the `${neo4j_dir}/conf/neo4j-server.properties` file.

`neo4j_server_port`

This integer value is the host IP port for the neo4j server. This value should match the `org.neo4j.server.webserver.port` parameter value in the `${neo4j_dir}/conf/neo4j-server.properties` file.

`slc_server_host`

This is the hostname of the inBloom API service. Currently, only the value "`api.sandbox.slcedu.org`" has been tested to work.

`slc_token_check_path`

This is the URL pathname inBloom "Access Token Check" service. Currently, only the value "`/api/rest/system/session/check`" has been tested to work.

`cache_type`

This is type of cache used. In memory "`local`" and "`memcached`" are the acceptable values, although `memcached` is the only useful caching mechanism. `local` is generally only used for debugging. If no value is set, then caching is turned off.

`use_cache`

This boolean value indicates if the caching system is turned on.

`memcached_host` and `memcached_port`

These should match your memcached configuration.

`bootstrap_filenames`

This list of strings should be set to the full pathnames of the bootstrap schema files. At present, the set of recommended schema files are:

- `${lri_dir}/lri_schema/bootstrap.json` -- Core bootstrap schema needed for inBloom Index operation
- `${lri_dir}/lri_schema/cc_schema.json` -- Learning Map Data Model schema
- `${lri_dir}/lri_schema/schema.org_fixed.json` -- a snapshot of the schema.org schema
- `${lri_dir}/lri_schema/lrmi_fixed.json` -- LRMI schema extensions to schema.org

`default_creator_guid`

This string indicates the default creator UID. It is generally only used for initial inBloom Index instance creation and regression testing.

`admin_access_tokens`

This name/value map provides the authentication mechanism for administrative writes through the inBloom Index servers' API. Each name is an access token (referred to by the `access_token` field of the `opts` parameters of HTTP requests to the API), and each corresponding value is the UID of the creator of the written assertions.

`delegate_tokens`

This is a list of access tokens that, when used in the `access_token` field, allow the writer to pass an `on_behalf_of` value in `opts`, which is then used at the creator UID.

NOTE: This is a very dangerous mechanism and should be used with care!

wsgi_config.json

The inBloom Index server uses UWSGI as its app platform to talk with the NGINX web server. While the specifics of this file are particular to UWSGI, a few of the fields are LRI specific:

wsgi-file

This string should be the full pathname to the `webapp.py` file in the main inBloom Index directory.

pyargv

This string should be the full pathname of the `lri_config.json` file.

Enumeration Guide

Data Representation of Enumerations in the LRI

Enumerations are fixed collections of values chosen in a declared value space. For example "male" and "female" are members of the "gender" enumeration. In the inBloom Index, enumerations are implemented as "member" entities of "enumeration" entities, not as literals chosen from a set. The enumeration entity schema currently are:

```
  "enumeration":{
    "comment_plain": "A finite, enumerated collection of entities",
    "id": "urn:lri:entity_type:enumeration",
    "label": "Enumeration",
    "specific_properties": [
        "urn:lri:property_type:has_enumeration_member"
     ],
    "supertypes": ["Thing"]
   },
  "enumeration_member":{
    "comment_plain": "A member of an enumeration entity",
    "id": "urn:lri:entity_type:enumeration_member",
    "label": "Enumeration Member",
    "specific_properties": [
       "urn:lri:property_type:is_member_of_enumeration"
     ],
    "supertypes": ["Thing"]
   }
```

Which refer to the properties:

```
    "urn:lri:property_type:is_member_of_enumeration": {
      "comment_plain": "This entity is a member of an enumeration
collection",
      "domains": ["urn:lri:entity_type:enumeration_member"],
      "id": "urn:lri:property_type:is_member_of_enumeration",
      "is_unique": false,
      "mandatory": false,
      "label": "Is Member Of",
      "is_primary": true,
      "ranges": ["urn:lri:entity_type:enumeration"],
      "reverse": "urn:lri:property_type:has_enumeration_member"
      },
    "urn:lri:property_type:has_enumeration_member": {
      "comment_plain": "This enumeration contains the member",
      "domains": ["urn:lri:entity_type:enumeration"],
```

```
      "id": "urn:lri:property_type:has_enumeration_member",
      "is_unique": false,
      "mandatory": false,
      "label": "Has Member",
      "is_primary": false,
      "ranges": ["urn:lri:entity_type:enumeration_member"],
      "reverse": "urn:lri:property_type:is_member_of_enumeration"
    }
```

As an example of enumerations, a draft of LRMI controlled vocabulary alignment and media type enumerations can be found in the `query_sequencer.py` input file:

```
${lri_dir}/test_suites/load/lrmi_enumerations.loadable.yaml
```

Query Sequencer Guide

The `query_sequencer.py` tool provides mechanisms for:

- Simple regression/unit tests of the inBloom Index API
- A bulk ingest or extraction tool.

Query sequencer files are generally YAML (but may be JSON) files containing a sequence of inBloom Index API queries to be executed. Currently, these are

Command line execution

This tool can be invoked with the syntax:

```
query_sequencer.py <LRI_HOST_IP:LRI_HOST_PORT> <SEQUENCE_FILENAME>
```

For example, the command used to load the Common Core State Standards would look something like:

```
${lri_dir}/query_sequencer.py 192.168.1.1:8000
$LRIDIR/test_suites/load/ccss_suite_new.yaml
```

Or to run a simple regression test:

```
${lrt_dir}/query_sequencer.py 192.168.1.1:8000
$LRIDIR/test_suites/regression/suite_01.yaml
```

Which would return output that looks like:

```
Running Step: step000
Running Step: step001
Running Step: step002
Running Step: step003
Done...
```

For every executed sequence, a report file is generated with same filename as the input sequence file, except that the file suffix is changed from `.yaml` to `.report.json`. e.g. For the above regression test, the output report filename would be:

```
$LRIDIR/test_suites/regression/suite_01.report.json
```

Existing Sequences

Examples of sequences included in this distribution can be found in:

- `$LRIDIR/test_suites/load` -- Bulk loadable data sets
- `$LRIDIR/test_suites/regression` -- Collection of inBloom Index API regression/unit tests

Schema Update Guide

This is a simple recipe for updating the schema for an inBloom Index instance.

Some things to know about how the inBloom Index handles schema:

- Schema are read at server startup time. Updates to schema after startup are not used until the next startup.
- When starting with a clean (empty) inBloom Index database, the `create_lri` command of the `lri_admin.py` tool is used to load the initial bootstrap schema. The bootstrap schema files are declared in the `bootstrap_filenames` of the `lri_config.json` file.
- Schema in the inBloom Index are represented as data entities/properties, just like any other data. To update schema, one performs inBloom Index write operations on the schema entities/properties. Writes to schema must be performed as an administrator, using one of the `admin_access_tokens` declared in the `lri_config.json` file.

Schema update recipe

1. Save a snapshot of the database using `lri_admin.py take_snapshot`
2. Perform the desired schema modifications to the live inBloom Index database using the inBloom Index API.
3. Use the `./lri_admin.py validate_live_schema` to verify that the schema changes retain a globally consistent state. If the validation fails, go to step #2.
4. Stop and restart the inBloom Index server to verify that the schema load properly. If it succeeds, the schema update is complete. If it fails, restore the database from the saved snapshot and start over from step #2.