Software Development on Linux Systems

4002-XXX-XX

By

Cody Van De Mark

This work is licensed under the Creative Commons Attribution-ShareAlike 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/3.0/ or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

Today

- Integrating with Services
- Databases
- Couchdb
- D-bus

Integrating with Services

- In open source software, none of the software functionality is hidden or obfuscated

- Since the code is open source and intended for others, the functionality is presented to the end user and they can modify it

- This makes software/system integration much easier and allows for direct integration

Integrating with Services

- In proprietary code, other developers can only integrate through APIs or difficult reverse engineering
- This limits the ability to integrate with propriety software, minimizing the system/software integration code available
- In open source, you can directly access the code you want to integrate with, and even modify it to your own needs

Integrating with Services

- Integration with open source software is prevalent as many open source projects have included integration code with a plethora of other projects
- Many integration tools are built into the Linux desktop and many more are available in the repositories
- To ease the process of integration further, many libraries exist for the sole purpose of allowing code integration
- Some libraries even allow integration across many languages

Integrating with Services

- As open source has many integration libraries and tools available, integration is usually much easier on Linux systems
- Most open source services have binding libraries available for many languages
- Binding libraries allow you to directly integrate with the service directly from your code
- On Linux, these libraries are available in the repositories and are configured for you

Databases

- In lab, you will integrate software with MySQL and SQLite
- In Linux, there is tremendous integration support for major open source databases
- You will find integration with these databases can be done through most major languages and communication is done for you

Databases

- In lab, you will connect to MySQL using c++
- This a more complicated solution, but most of the management is done for you
- After you have downloaded the MySQL c++ libraries, you can leverage your MySQL database simply my importing mysql++.h

Databases

- After you have imported the MySQL header file, you can directly connect with

  mysqlpp::Connection conn(“dbName”, “address”, “user”, “pass”);

- Queries can be written and executed with

  mysqlpp:Query query = conn.query(“SELECT * FROM table”);
  mysqlpp:StoreQueryResult res = query.store();

  This will return your results in an indexed associative array

  IE: res[index][“column name”]

Databases

- Insert statements are written similarly to queries

  Example: ***Don't use this in production as it is vulnerable to sql injection**

```
string idx = “1”;
string text = “\”Hello World\””;
string date = “now()”;


mysqlpp:Connection conn(“dbName”, “address”, “user”, “pass”);
mysqlpp:Query query = conn.query();


query << “INSERT INTO messages (id, message, date) VALUES
(“ << idx << “, ” << text << “, ” << date << “);”;


mysqlpp::StoreQueryResults res = query.store();
```

Datebases

- As you can see MySQL integration in c++ is not overly complex as most of the work has been done for you
- It is even easier in Python and other languages, though the approach is similar

Databases

- In lab, you will use another popular open source database that is quite a bit different
- SQLite is a database designed for individual instances and is usually used on a per application basis
- SQLite is a relational database that is designed to be embedded within an application.
- The SQLite database will run within the connecting client

Databases

- SQLite is often used for user and application data
- This allows applications to store preferences, user data and run time data in a relational database
- You may use SQLite as an object store for character information in a game or user bookmarks in a web browser, etc
- SQLite is used to fit the storage needs of any application to allow you to store information while the software is running and to store information while the application is not running

Databases

- We will look at SQLite integration on Linux, this time using Python
- A Database connection in Python is very simple compared to the c++, though follows the same process
- To use SQLite in Python, you just need to import sqlite3

Databases

- Once you have imported the SQLite package in Python, you can connect to an existing or new database with

  ```
  conn = sqlite3.connect('/path/to/database/file')
  ```

- You will need a cursor, which is a pointer to your database;

  This allows you to make any changes to your cursor without committing them to the database, then you may commit them all at once

  ```
  cursor = conn.cursor()
  ```

Databases

- After the cursor has been made, you may execute any queries before you commit the actual writes to the database

  Example: *Again, direct queries are not secure

```
conn = sqlite3.connect('/path/to/database/file')
cursor = conn.cursor()

cursor.execute('''CREATE TABLE names (ID INT primary key, name TEXT)''')

cursor.execute('''INSERT INTO names VALUES(1, “person1”) ''')

conn.commit()
```

Databases

- For security, you can use question marks and pass the value in;

  This prevents SQL injection

  Example:

```
datalist = (2, “person2”)

cursor.execute('INSERT INTO pictures VALUES(?, ?)', datalist)

 conn.commit()
```

Databases

- Querying results can also be done with the cursor

- The cursor allows us to make one query and save the results

  Example:

  ```
  cursor.execute('''SELECT * FROM names''')

  for row in cursor:
      print row
  ```

Databases

- After you are done, you should close your connection with

  ```
  cursor.close()
  conn.close()
  ```

- You may also create temporary tables two ways
  - On hard drive with

    ```
    conn = sqlite3.connect('/tmp/media')
    ```

  - In RAM only with

    ```
    conn = sqlite3.connect(':memory:')
    ```

CouchDB

- CouchDB is also a database, but is being discussed in a different section because it is very different from traditional databases
- CouchDB is a document store, key-value database;

  This means it is not a column-row database
- CouchDB is one of the NoSQL databases, meaning it does not follow SQL syntax

CouchDB

- CouchDB is a key value database, meaning it stores a key which an attached value in the form of a JSON object

  Example:

```
Key          :          Value

wiki         :          {
                            “name”:     “Open Source”,
                            “url”:      “http://ist.rit.edu/open/”,
                            “views”:    1001
                        }
```

CouchDB

- With CouchDB, you store JSON documents

- Ideally, the “keys” will be unique in a key value database so that can query each document by unique name

  IE: doc.id = “wiki”

- You can also give documents attributes, such as “type” ;

  This way you can do things like (if doc.type == “customer”)

CouchDB

- Since CouchDB is very different than a relational database, it may be hard to grasp
- A key is vaguely similar to a table name and should be unique, but keys can be data other than a tables name, such as a variable type
- A value is vaguely similar to a table with column
- They should <u>not</u> be thought of as tables and table names, but may be easier to grasp that way

CouchDB

- Each key is a unique value you can query, while each value is a list of attributes with their corresponding values

- Where these differ is you can query many documents at once using an attribute, such as type:

  By this it means, you can get the name and address attributes out of all documents whose type is “customer”

CouchDB

- CouchDB is designed around the map-reduce function
- The 'map' function is the idea of finding (mapping) data, such as the appearance of a word
- The 'reduce' function is the idea of totaling (reducing) the found data, such as the total number of occurrences of a word

CouchDB

- Example of map-reduce functionality

```
#!/usr/bin/env python
mapped = []
d = "This is a bunch of words in a doc in a doc"

def mapd(document):
        words = document.split()  #gives us each word in array
        for word in words:
                mapped.append("%s %s" % (word, 1))

def reduce(findword, mapped):
        total = 0
        for line in mapped:
                cur_word, num_found = line.split()
                count = int(num_found)

                if cur_word == findword:
                total += count

        print total
```

CouchDB

- CouchDB is installed as a service on your machine, similar to how a database is a running service
- The default URL for CouchDB is http://localhost:5984/
- You can change the port and address in the CouchDB config file /etc/couchdb/default.ini
- CouchDB can be integrated with directly through curl or through a number of languages

CouchDB

- With curl you can create a new document with PUT

```
curl  -X  PUT  http://localhost:5984/newDB
```

- You can retrieve the information from your document with GET

```
curl  -X  GET  http://localhost:5984/newDB
```

- You can also delete documents with DELETE

```
curl  -X  DELETE  http://localhost:5984/newDB
```

CouchDB

- You may also add documents with PUT

```
curl -X PUT    http://localhost:5984/newDB/newDoc
               -H  “Content-Type: application/json”
               -d   '{ “id”:1, “name”:“doc1”}'
```

- Curl allows you to interact with CouchDB similarly to how the MySQL command line allows you to interact with MySQL

CouchDB

- You should be cautious of CouchDB and curl, however, as CouchDB is not checking security by default allowing curl to interact from anywhere by anyone

- CouchDB can be run securely by setting up security settings or by only allowing localhost access

- You need to import a couple items to use CouchDB in Python:

  ```
  import couchdb
  from couchdb.client import Server
  ```

CouchDB

- You can connect to your CouchDB service with

```
server = Server('http://127.0.0.1:5984/')
```

- You can create a database with

```
db = server.create('newDB')
```

- You may also delete a database with

```
del server['newDB']
```

CouchDB

- If you have already created a database, you can create a document (key) and add your values

```
db['person1'] = dict(id=1, name='myName', email='person@rit.edu')
```

- Queries are done through javascript that is run against the json object

```
queryString = '''function(doc)
                  {
                      if (doc._id == “person1”) {
                              emit(doc._id, [doc.name, doc.email]);
                      }
                  }'''

results = db.query(queryString)
```

CouchDB

- Results are stored by their key and value

```
results = db.query(queryString)

for result in results:
        print “Key used: ”, result.key, “\tValue: ”, result.value
```

- You may also add or modify values in an existing document

```
doc = db.get('person1')            #gets person1 document
doc['name'] = 'Nick'               #Changes name to Nick
doc['id'] = int(doc['id']) + 1     #Parses ID as int and adds 1
doc['age'] = 24                    #Creates new value, age
```

D-Bus

- Another service and integration tool is Desktop Bus (D-Bus)
- D-Bus is an open source inter-process communication system
- D-Bus allows your software to offer services by exporting functions to D-Bus as a service
- D-Bus allows processes to call each other through a communication layer

D-Bus

- The language of a program does not matter as long as they can connect to D-Bus
- This allows any language to call functions on D-Bus regardless of which language they are written in
- The called function will still receive parameters (if applicable) and return a value (if applicable) as normal

D-Bus

- D-Bus is split into two components: Services and Clients
- Services are programs that are offering specific methods as services on D-Bus, though they can be clients as well
- Clients are programs that connect to D-Bus to leverage existing software and libraries, but do not offer their own services

D-Bus

- D-Bus has two actively running buses you can utilize:

System Bus	-	System wide processes communication bus
Session Bus	-	Login session bus (user specific session)

- The system bus will have system level processes on it, such as printers

- The session bus will have user specific processes on it, such as Banshee media player, Skype, etc

D-Bus

- Services need to import several items:

```
import dbus
import dbus.service
import gobject
from dbus.mainloop.glib import DbusGMainLoop
```

- Clients only need to import the main dbus package

```
import dbus
```

D-Bus

- To create a D-Bus service, you need to create a class you can export

```
class MyService(dbus.service.Object):
        def __init__(self):
                bus_name = dbus.service.BusName('org.my.service', bus=dbus.SessionBus())
                dbus.service.Object.__init__(self, bus_name, '/org/my/service')

        @dbus.service.method('org.my.service')
        def hello(self):
                return “Hello”

        @dbus.service.method('org.my.service')
        def goodbye(self):
                return “Goodbye”

DbusGMainLoop(set_as_default=True)
my_service = MyService()
loop = gobject.MainLoop()
loop.run()
```

D-Bus

- To create a client, you just need to grab an object of the service from the bus

```
bus = dbus.SessionBus()
remote = bus.get_object(“org.freedesktop.DBus”, “/org/freedesktop/DBus”)

iface = dbus.Interface(remote, 'org.freedesktop.DBus')

testService = bus.get_object('org.my.service', '/org/my/service')

print testService.hello()
print testService.goodbye()
```