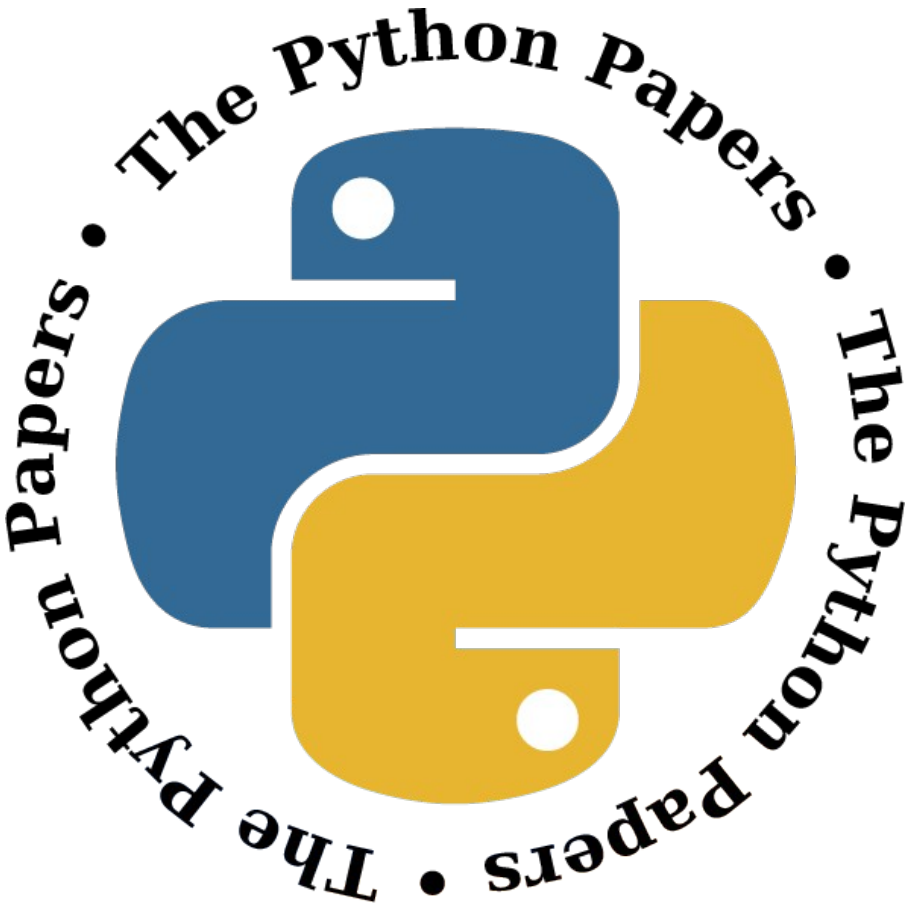
The Python Papers



Volume 2, Issue 1
pythonpapers.org

Journal Information

The Python Papers

ISSN: 1834-3147

Editors

Tennessee Leeuwenburg
Maurice Ling
Richard Jones

Referencing Information

Articles from this edition of this journal may be referenced as below:

Author, “Title” (2007) *The Python Papers*, Volume N, Issue M, pp. m:n

e.g. Maurice Ling, “Firebird Database Backup by Serialized Database Table Dump”, (2007) *The Python Papers*, Volume 2, Issue 1, pp. 7:15.

Copyright Statement

© Copyright 2007 The Python Papers and the individual authors
This work is copyright under the Creative Commons 2.5 license subject to Attribution, Noncommercial and Share-Alike conditions. The full legal code may be found at http://creativecommons.org/licenses/by-ncsa/2.1/au/

The Python Papers was first published in 2006 in Melbourne, Australia.

Referees

1. Tennessee Leeuwenburg (tennessee@tennessee.id.au)
2. Helen Borrie, Firebird Project administrator and author of "The Firebird Book: A Reference for Database Developers" (Apress, 2004)
3. Ann Harrison, Firebird Project administrator

The Python Papers

Volume Two, Issue One : February 2007

In this Edition:

Articles and Editorials:

Peer Reviewed Submissions:

Letter from the Editor

G'day Pythonistas! Welcome to Issue Two of *The Python Papers.* It has been an exciting time and we are pleased to have reached this milestone.

The last few months have involved a lot of effort connecting with many groups within the Python community. I'd like to say a big hello to all the people who have provided their input in making this a reality: the *python-advocacy* list, *comp.lang.python*, the Python User Groups that responded to the call to participate and also many individuals.

The effort to support and assist in the growth of the Python community is an ongoing one. One way to help in this process would be to build a strong presence of Python developers in various online networks. One that springs to mind is LinkedIn[1], a job-related social networking site.

If we could encourage Python developers to start adding each-other to their LinkedIn network, then we should be able to create a well-connected developer network with business and industry contacts. This should benefit everyone -- both people looking for Python developers, and also people looking for work.

This effort has already begun online, with many from the *python-advocacy* list linking to each-other, and word is also leaking into the blogosphere[2].

The idea is that Python developers link to other people they know personally (including via online groups). Just a few such connections could put more Python developers within a few degrees of separation of a huge network of employers and social contacts.

The benefits of this are manifold: each individual is connected to more people who might be able to help them in future, both from the client and employer side; Python developers generally become a well-connected resource, making Python more visible to the community at large and potentially helping to drive its acceptance and adoption.

This is also the first issue where we have attempted to publish both a PDF and an HTML edition. The choice of format was clearly an issue for many people on both sides. Going forward, we will continue to use PDF as our primary 'authorative' version for the purposes of page numbering and referencing, however the HTML editions will be made public in order to reach a wider audience.

Stop Press – Contribute and win!

We are also now able to announce that we have a free license to Wing IDE, a fantastic Python development environment created by Wingware[3]. In order to win a 3-OS license for this software, all you have to do is contribute an article[4] to the next edition of The Python Papers. Keep your eyes out next edition to see if you are the lucky winner!

Introducing The Team

Tennessee Leeuwenburg – Editor-in-Chief

Tennessee Leeuwenburg is a software developer working at the Australian Bureau of Meteorology on automatic text generation of weather forecasts. Prior to this he spent time working on meteorological data transfer standards in the form of the OpenDAP database system.

1 http://www.linkedin.com/
2 http://agiletesting.blogspot.com/2007/01/connecting-to-people-on-linkedin.html
3 http://wingide.com/
4 Contributors must not be on the editorial board to be eligible

Maurice Ling – Associate Editor

Maurice Ling is a PhD candidate in the department of Zoology of the University of Melbourne working on text analysis of biological literature for the purpose of understanding hormone interactions in the mouse mammary cell.

Richard Jones – Associate Editor

Richard is Common Ground's[5] senior software developer with 10 years of broad experience working with multiple languages and tools in web-based management systems, data archive, meta-data systems, business systems, e-commerce and communications. He also runs the bi-annual PyWeek Game Programming Challenge[6] and is involved with the organisation of the Open Source Developer's Conference[7] in Melbourne, Australia.

Stephanie Chong – Design Assistant

One of Stephanie's first comments regarding the The Python Papers was “how can we make it prettier?” We hired her on the spot.

Stephanie is currently studying Arts/Law at the University of Melbourne.

Contacting The Python Papers

The editors may be contacted via email at:

editor@pythonpapers.org

We are always looking for new writers for *The Python Papers.* If you would like to contribute an opinion piece, an article, participate in an interview or submit a paper for review and publication, please don't hesitate to contact us.

5 http://commongroundgroup.com/
6 http://www.pyweek.org/
7 http://osdc.com.au/

Python Interview – Ron Stephens of Python 411

The Python Papers (TPP) talks to Ron Stephens of *Python 411*[8].

TPP: Who is Ron Stephens?

Ron> *I am a middle aged man living in Connecticut, USA, a father of four children and an electronics industry executive who discovered Python while attending a Linux Users Group with my son about 6 years ago.*

TPP: What motivated you to start the Awaretek site?

Python Learning Foundation

1. An organisation promoting Python information

2. Host of the popular "Python 411" podcast series

Text 1: Source: www.awaretek.com

Ron> *I actually started the AwareTek site about 10 years ago as a means of self-expression. The Python orientation of the site was inspired when, not long after I discovered Python, I read a book about the wonderful open source community in which Guido van Rossum was quoted talking about "Computer Programming for Everybody". I felt an overwhelming desire to participate.*

TPP: Have you made many new contacts since starting to publish information?

Ron> *Yes, I get many emails from Pythonistas from all over the world, as well as talking to the folks whom I interview for the Python411 podcast series.*

TPP: What do you think is the current biggest problem with Python (in a broad sense)?

Ron> *You know, I don't see any problems, I'm a glass-half-full kind of guy. I can hardly believe what a beautiful language and powerful set of tools has been created by a group of dedicated programmers and given freely to all of us to use, and all of this is out of the goodness of people's hearts. It doesn't get much better than this.*

TPP: What is best best feature of Python (again in a broad sense)?

Ron> *Ease of learning and ease of use are the killer features of Python.*

TPP: As a contributor to the Python community, do you feel well connected to other Python developers?

Ron> *I attended PyCon last year for the first time, and I plan on being in Dallas in late February to do so again, and that is my first real face to face exposure to other Pythonistas. I highly recommend PyCon to anyone who has a chance to go. I do not a have local user's group close enough to attend, so other than PyCon, I do miss talking with real live Pythonistas; but I am thankful for the online virtual world, without which the Python and larger open source communities as we know them could not exist.*

TPP: What are some of the most exciting Python projects that you know about?

Ron> *My favorite Python projects include 'Chandler', a sort of cross platform uber-PIM; 'PythonCard', an incredibly easy to use GUI creator; and 'Firedrop', a weblog and content management system for Python hackers.*

8 http://www.awaretek.com/

Programming Idioms pt 2 -- Design Patterns

Tennessee Leeuwenburg

I'll admit it – I live under a rock. I hadn't heard about Design Patterns until a couple of months ago. So sue me. That said, I'll bet I'm not the only one. So what's a Design Pattern?

Simple. It's boilerplate design. The idea seems simple enough – many problems can be expressed in common ways. If that's true, then there should exist some small set of program structures which are useful for many, many problems.

Identifying these and documenting them is what is involved in creating a Design Pattern. Armed with a library of Design Patterns, solving a complex problem can be simplified by expressing it in terms of a problem that you *already* know how to solve – at least more or less.

The seminal text in this area is *Design Patterns: Elements of Reusable Object-Oriented Software*[9], often referred to as the *GoF* or *Gang of Four*, partly because it has four authors and partly because it is a much shorter title.

One of the first things that occurred to me when I was thinking about this topic was *MapReduce*[10], an algorithm used by Google in distributed processing.

Many problems can be expressed in terms of operations which also happen to be easily run in parallel. If a problem can be expressed in terms of a Map() and then Reduce() function, then it is easy to make use of distributed processing.

While MapReduce is not one of the patterns addressed in *GoF*, many that are may already be familiar to many programmers. These include Factories, Singleton classes, Adapters, Iterators and others.

They have all been built many times over because they are a sensible way to deconstruct a program design. Design Patterns are an attempt to bring additional tools born from experience to bear on problem analysis and code design.

There are further similarities with, for example, Agent-Oriented architectures and other such design systems. If they have anything in common, it is because there is an overriding truth that a complex problem must often be broken into component parts before a solution may be found.

A programming idiom is the lowest-level expression of this truth. As discussed in part one of this series, a programming idiom is a practise adopted to help make the meaning of code clearer.

It is perhaps reasonable to say then that a Design Pattern is a practise adopted to help make the medium-scale structure of code clearer.

Design Patterns for Python

Many Design Patterns appear to be closely linked to particular Object-Oriented models which do not apply to Python as neatly as to other languages. For example, Python's dynamic duck typing obviates some of the patterns which relate to issues caused by static typing, for example the Abstract Factory and Builder Patterns.

For specific information on how to use design patterns in your own code, “Design Patterns in Python”[11] is a good start. However, I will briefly list the Design Patterns which appear to me to be most relevant to Python. These patterns are, in no particular order: Prototype; Singleton (optionally with Borg style); Composite; Facade; Flyweight; Proxy (*looks Evil, Ed*); Chain of

9 http://www.amazon.com/Design-Patterns-Object-Oriented-Addison-Wesley-Professional/dp/0201633612
10 http://labs.google.com/papers/mapreduce.html
11 http://www.python.org/workshops/1997-10/proceedings/savikko.html

Responsibility; Command; Iterator; Interpreter; Mediator; Memento; Observer (or Listener) and Strategy.

Having read up on each of these (see GoF or else Wikipedia[12] has an entry for each, see “Gang of Four”), I made a sub-list of patterns that I have used myself: Strategy; Observer (or Listener); Iterator; Chain of Responsibility; Facade; Composite and Singleton.

However, I have rarely used these patterns as part of an overarching design strategy. Indeed, it is still unclear how these patterns may be best integrated into system design for significant components or applications. They appear more applicable to small-scale problems.

It is not clear, for example, how the Iterator pattern applies to a larger task – implementing a Customer Relations Manager for example. Only once that larger system is broken down to include a component which processes each customer does an Iterator become useful in solving the problem.

Design Patterns, then, are a higher level of abstraction than an algorithm, but a lower level of abstraction than system design.

When to use Design Patterns

The question then is, 'When should a Design Pattern be used?' If not during system design (too low-level) and not line-by-line (too high-level), then when?

There are no hard and fast rules. Little academic research has been done on the topic, and there are no de-facto standards here. As such, if you see an opportunity to make use of a Design Pattern, doing so may well save you much time spent in the design phase.

Typically, the right time is when working on an individual component of a larger system. For example, supposing the task at hand were to build an Internet chat room. This might require a central server to which multiple clients could connect, some kind of user interface for interacting with the system, and messaging processing logic on both sides. This is the system design level.

Let's dive into the server logic. The server will need to manage client connections and handle the relevant message-passing. A simple server could be structured around a simple main loop (Iterator pattern):

```
class ChatServer:

    def _mainLoop(self):
        outgoing = ''

        # Get messages coming in from each client
        for client in clients:
            incoming = client.getMessages()
            outgoing = outgoing + '\n'.join(incoming)

        # Push the message stack back to each client
        for client in clients:
            client.recieveMessages(outgoing)
```

Here we see the use of an iterator which is central to the Python language. The ability of *for* loops to process interated information is very useful. It allows iteration over lists, sequences or even user-defined functions.

The Iterator pattern can also be used from the other side, so to speak, in writing code which generates the items to be iterated over. It could be used, for example, as the basis of an event-

12 www.wikipedia.org

based system. Writing an Iterator which *yields* events would allow a simple *for* loop to control program execution.

It can be seen that there are uses for Design Patterns just below the level of system design – when building components identified during this earlier phase. This process of taking a design requirement, and looking for a Design Pattern which might be able to meet that requirement, is not just an exercise for its own sake. Rather, it is a logical short-cut. Implementing a problem in an already familiar way will make the job quicker and simpler. Should the code ever be passed to another programmer, they will be able to understand the execution flow that much more easily.

To use another analogy, using Design Patterns liberally will encourage the use of common structures in various system components, indeed across programs and even across programming languages and systems. While a Java implementation may not be syntactically the same as a Python implementation, the underlying structure of the code will be similar.

Python User Group Highlights

This regular section will cover the activities of Python User Groups (PUGs) and Special Interest Groups (SIGs) around the Planet. If your local group would like to feature in the next edition, please contact editor@pythonpapers.org. This issue we cover the Toronto Python User's Group and Python Brasil. *The Python Papers (TPP)* spoke to Mike Fletcher and Rodrigo Senra.

Toronto Python User's Group: PyGTA

Mike Fletcher> *PyGTA is the Toronto Python User's Group. It was first convened in September of 2002 and meets monthly at the Linux Caffe in downtown Toronto. We have around 8 regular attendees, with the more popular discussions topping 30 attendees. Our members have a larger than normal (for Python) representation in the embedded systems space, but the range of interest is broad, from accounting and billing software through 3D graphics, web development, systems administration, media editing and radio broadcasting.*

Illustration 1: PyGTA meeting in Jan 2007

Python is used by a considerable number of companies in the Greater Toronto Area as an in-house development language. We also have a number of independent contracting companies who provide bespoke software development. Although we know there are a lot of academic and personal Python programmers in the city, we don't seem to attract them to the meetings as much as the professional programmers, a situation we would love to rectify.

HTH,
Mike

Contact details for reaching Mike and others at PyGTA are available at the PyGTA website.[13]

13 http://web.engcorp.com/pygta

PythonBrasil

TPP: Where is the user group based?

Rodrigo Senra> *There is no basecamp. We are scattered across the country. But our virtual headquarters is our Wiki portal.*[14]

TPP: Who are the active members?

Rodrigo> *We keep a detailed list by state at with about a 100 people. These are the most active. In our mailing list we have over 700 subscriptions.*[15]

TPP: How long has the group existed for?

Rodrigo> *It is about 4 or 5 years old.*

Illustration 2: Pycon Brasil 2006. These are some of the usual suspects, including world famous figures like: Sidnei da Silva, Dorneles Tremea, Gustavo Niemeyer

TPP: How frequently are meetings held?

Rodrigo> *Next year we are going to have our 3rd national conference about Python/Zope/Plone called... PyconBrasil ;o)*

The first one was in 2005 with an audience around 100 people. This year the crowd has tripled.

TPP: What are the particular programming interests held by members?

Rodrigo> *It is a big country, and interests are vast.*

TPP: Are there any upcoming events?

Rodrigo> *August 2007 -> 3rd PyConBrasil. We' ll announce it at c.l.py*[16] *in due time.*

TPP: How is Python used in your country?

Rodrigo> *I have presented "The growth of the Brazilian Python Community" during Europython 2005. Thit is the tip of the iceberg. I can make the slides available in case there is any interest.*

14 http://www.pythonbrasil.com.br

15 http://www.pythonbrasil.com.br/moin.cgi/GrupoDeUsuarios

16 http://groups.google.com/group/comp.lang.python

Firebird Database Backup by Serialized Database Table Dump

Maurice HT Ling
Department of Zoology, The University of Melbourne, Australia

Correspondence: mauriceling@acm.org

Abstract

This paper presents a simple data dump and load utility for Firebird databases which mimics *mysqldump* in MySQL. This utility, *fb_dump* and *fb_load*, for dumping and loading respectively, retrieves each database table using *kinterbasdb* and serializes the data using *marshal* module. This utility has two advantages over the standard Firebird database backup utility, *gbak*. Firstly, it is able to backup and restore single database tables which might help to recover corrupted databases. Secondly, the output is in text-coded format (from *marshal* module) making it more resilient than a compressed text backup, as in the case of using *gbak*.

Categories and Subject Descriptors
H.2.7 [**Database Administration**]: Logging and Recovery
E.5 [**Files**]: Backup/recovery

1. Introduction

The only database backup and restore utility in Firebird database management system is *gbak* (chapter 38 of Borrie, 2004), which required the database to be in an uncorrupted state and backed-up into a compressed textual format known as External Data Representation (XDR) format (Srinivasan, 1995). Although some forms of database corruption can be repaired using the standard housekeeping tool, *gfix* (chapter 39 of Borrie, 2004), serious errors can still persist, preventing standard backup to be performed (page 936 of Borrie, 2004). In such cases, data within the corrupted database might be rescued by exporting the data out by third-party tools or *qli* within the standard installation.

On the other hand, MySQL's (another popular open source RDBMS) backup utility is a database dumper utility which dumps the database, table by table, as SQL insert statements. Although there are advantages with Firebird *gbak* utility (for a complete discussion of *gbak*'s capabilities, please refer to chapter 38 of Borrie, 2004), it might be useful to be able to backup or dump individual Firebird database tables into a text or text-coded format as requested by several users (Palaparambil, 2006; Piridi, 2006). A major advantage of text or text-coded format over binarycompressed text format is the resilience of text format over compressed format to corruption. Corruption in text or text-coded format may result in loss of several records but corruption in compressed format may result in loss of entire tables.

This paper presents a simple data dump and load utility for Firebird databases (versions supported by kinterbasdb library which included virtually all versions of Firebird) which mimics *mysqldump* in MySQL. Each Firebird table is retrieved using *kinterbasdb* library and serialized into a text-coded file using Python's *marshal* module.

Two other third-party backup systems available for Firebird databases are nBackup (Vinkenoog, 2005) and dBak (http://www.telesiscomputing.com.au/dbk.htm) which were presented as alternatives to *gbak*. Both dBak and nBackup are comparable to the proposed method in the aspect of table-level data extraction. However, dBak relied on Borland Delphi's IBObjects; hence, unlike the proposed Python-based method, dBak is not available for Unix-based systems. Despite not being able to ascertain whether nBackup will be available for both Unix-based systems and Microsoft systems, nBackup is only available for Firebird version 2.0 and above as a standard Firebird utility (Vinkenoog, 2005). The proposed method in this paper used *kinterbasdb* library;

thus, making it available for use with all *kinterbasdb*-supported versions of Firebird which included versions lesser than 2.0. The version of Firebird used in this study is version 1.5.2. Although the proposed method in this paper lacked the ability to create incremental backups with similar ease as that of nBackup, it is able to backup multifile databases and change database owner by re-creating a clean database. Therefore, we consider our proposed method as another possible complementary Firebird database backup method (utility) to *gbak*, dBak, and nBackup.

2. Implementation

```
"""
Firebird database dumper which mimics MySQL database dump. The resulting files are serialized records.
Date created: October 18, 2006
Copyright (c) 2006, Maurice Ling
"""
import kinterbasdb, marshal, os, gc

db_path = 'employe2.fdb'
db_user = 'SYSDBA'
db_password = 'masterkey'
number_of_records = 500000      # number of records (tuple) in a dump file

sql_script = [
            ["cross_rate",
              "select from_currency, to_currency, conv_rate, cast(update_date as char(24)) from cross_rate",
              "insert into cross_rate (from_currency, to_currency, conv_rate, update_date) values (?,?,?,?)"]
           ]

con = kinterbasdb.connect(dsn=db_path, user=db_user, password=db_password)
cur = con.cursor()

data = {}

for table in sql_script:
    data.clear()
    gc.collect()
    data['insert_sql'] = table[2]
    print 'Dumping table: ' + table[0]
    cur.execute(table[1])
    rowcount, setcount = 0, 0
    temp = []
    for row in cur:
        temp.append(row)
        rowcount = rowcount + 1
        if (rowcount % number_of_records) == 0:
            setcount = setcount + 1
            data['data'] = temp
            temp = []
            f = open(table[0] + '.' + str(setcount) + '.dump', 'w')
            marshal.dump(data, f)
            print '%s records dumped into %s file' % (str(number_of_records), table[0] + '.' + str(setcount) + '.dump')
            f.close()
    if len(temp) > 0:
        setcount = setcount + 1
        data['data'] = temp
        temp = []
        f = open(table[0] + '.' + str(setcount) + '.dump', 'w')
        marshal.dump(data, f)
        print '%s records dumped into %s file' % (str(len(data['data'])), table[0] + '.' + str(setcount) + '.dump')
        f.close()
    print 'Total number of records in %s table: %s' % (table[0], str(rowcount))
    print 'Dumping ' + table[0] + ' table successful.'

"""
Firebird database loader to load dump files from fb_dump
Date created: October 18, 2006
Copyright (c) 2006, Maurice Ling
"""
import kinterbasdb, marshal, os, gc, sys

def load(f, db_path, db_user, db_password):
```

```
    con = kinterbasdb.connect(dsn=db_path, user=db_user, password=db_password)
    cur = con.cursor()
    data = marshal.load(open(f, 'r'))
    for tuple in data['data']:
        try:
            cur.execute(data['insert_sql'], tuple)
            con.commit()
        except: 'Unable to load record: %s' % tuple
    print '%s file loaded successful.' % f
    con.close()

if __name__ == "__main__":
    if len(sys.argv) < 5:
        print "Usage: python db_load.py <path_to_database> <database_user> <database_password> <dump_file>+"
    for dumpfile in sys.argv[4:]:
        load(dumpfile, sys.argv[1], sys.argv[2], sys.argv[3])
```

3. Evaluation

Two evaluation tests were carried out – the first evaluation demonstrated a proof of concept while the second evaluation demonstrated that the utility could be used on a larger database with reasonable performance.

In the first evaluation, a sample database was constructed using the following script as supplied as an example in the Firebird installation:

```
set sql dialect 1;
create database "employe2.fdb";
CREATE TABLE cross_rate(
    from_currency    VARCHAR(10) NOT NULL,
    to_currency      VARCHAR(10) NOT NULL,
    conv_rate        FLOAT NOT NULL,
    update_date      DATE,
    PRIMARY KEY (from_currency, to_currency));

INSERT INTO cross_rate VALUES ('Dollar', 'CdnDlr',  1.3273,  '11/22/93');
INSERT INTO cross_rate VALUES ('Dollar', 'FFranc',  5.9193,  '11/22/93');
INSERT INTO cross_rate VALUES ('Dollar', 'D-Mark',  1.7038,  '11/22/93');
INSERT INTO cross_rate VALUES ('Dollar', 'Lira',    1680.0,  '11/22/93');
INSERT INTO cross_rate VALUES ('Dollar', 'Yen',     108.43,  '11/22/93');
INSERT INTO cross_rate VALUES ('Dollar', 'Guilder', 1.9115,  '11/22/93');
INSERT INTO cross_rate VALUES ('Dollar', 'SFranc',  1.4945,  '11/22/93');
INSERT INTO cross_rate VALUES ('Dollar', 'Pound',    0.67774, '11/22/93');
INSERT INTO cross_rate VALUES ('Pound',  'FFranc',  8.734,   '11/22/93');
INSERT INTO cross_rate VALUES ('Pound',  'Yen',     159.99,  '11/22/93');
INSERT INTO cross_rate VALUES ('Yen',    'Pound',   0.00625, '11/22/93');
INSERT INTO cross_rate VALUES ('CdnDlr', 'Dollar',  0.75341, '11/22/93');
INSERT INTO cross_rate VALUES ('CdnDlr', 'FFranc',  4.4597,  '11/22/93');
```

The resulting serialized dump file is as follows:

```
{t^D^@^@^@data[^N^@^@^@(^D^@^@^@s^F^@^@^@Dollars^F^@^@^@CdnDlrf^Q1.3272999525070
19s^X^@^@^@1993-11-22 00:00:00.0000(^D^@^@^@s^F^@^@^@Dollars^F^@^@^@FFrancf^R5.9
1930000793457031s^X^@^@^@1993-11-22 00:00:00.0000(^D^@^@^@s^F^@^@^@Dollars^F^@^@
^@D-Markf^R1.7037999629974365s^X^@^@^@1993-11-22 00:00:00.0000(^D^@^@^@s^F^@^@^@
Dollars^D^@^@^@Liraf^F1680.0s^X^@^@^@1993-11-22 00:00:00.0000(^D^@^@^@s^F^@^@^@D
ollars^C^@^@^@Yenf^R108.43000030517578s^X^@^@^@1993-11-22 00:00:00.0000(^D^@^@^@
s^F^@^@^@Dollars^G^@^@^@Guilderf^R1.9114999771118164s^X^@^@^@1993-11-22 00:00:00
.0000(^D^@^@^@s^F^@^@^@Dollars^F^@^@^@SFrancf^R1.4945000410079956s^X^@^@^@1993-1
1-22 00:00:00.0000(^D^@^@^@s^F^@^@^@Dollars^E^@^@^@Poundf^S0.67773997783660889s
^X^@^@^@1993-11-22 00:00:00.0000(^D^@^@^@s^E^@^@^@Pounds^F^@^@^@FFrancf^R8.73400
02059936523s^X^@^@^@1993-11-22 00:00:00.0000(^D^@^@^@s^E^@^@^@Pounds^C^@^@^@Yenf
^R159.99000549316406s^X^@^@^@1993-11-22 00:00:00.0000(^D^@^@^@s^C^@^@^@Yens^E^@
^@^@Poundf^U0.006250000931322575s^X^@^@^@1993-11-22 00:00:00.0000(^D^@^@^@s^F^@
^@^@CdnDlrs^F^@^@^@Dollarf^R0.7534099817276001s^X^@^@^@1993-11-22 00:00:00.0000(
^D^@^@^@s^F^@^@^@CdnDlrs^F^@^@^@FFrancf^R4.4597001075744629s^X^@^@^@1993-11-22 0
0:00:00.0000t
^@^@^@insert_sqls\^@^@^@insert into cross_rate (from_currency, to_currency, conv
_rate, update_date) values (?,?,?,?)0
```

The resulting serialized dump file yielded the correct corresponding dictionary when de-serialized:

```
>>> import marshal
>>> data = marshal.load(open('cross_rate.1.dump', 'r'))
>>> data
{'insert_sql': 'insert into cross_rate (from_currency, to_currency, conv_rate, update_date) values (?,?,?,?)', 'data':
[('Dollar', 'CdnDlr', 1.327299952507019, '1993-11-22 00:00:00.0000'), ('Dollar', 'FFranc', 5.9193000793457031, '1993-
11-22 00:00:00.0000'), ('Dollar', 'D-Mark', 1.7037999629974365, '1993-11-22 00:00:00.0000'), ('Dollar', 'Lira', 1680.0,
'1993-11-22 00:00:00.0000'), ('Dollar', 'Yen', 108.43000030517578, '1993-11-22 00:00:00.0000'), ('Dollar', 'Guilder',
1.9114999771118164, '1993-11-22 00:00:00.0000'), ('Dollar', 'SFranc', 1.4945000410079956, '1993-11-22
00:00:00.0000'), ('Dollar', 'Pound', 0.67773997783660889, '1993-11-22 00:00:00.0000'), ('Pound', 'FFranc',
8.7340002059936523, '1993-11-22 00:00:00.0000'), ('Pound', 'Yen', 159.99000549316406, '1993-11-22
00:00:00.0000'), ('Yen', 'Pound', 0.0062500000931322575, '1993-11-22 00:00:00.0000'), ('CdnDlr', 'Dollar',
0.7534099817276001, '1993-11-22 00:00:00.0000'), ('CdnDlr', 'FFranc', 4.4597001075744629, '1993-11-22
00:00:00.0000')]}
```

The second evaluation test was performed on a research database containing more than 50 tables of which 48 million records in 20 tables were chosen for backup. This task took about 22 minutes to complete on a 2GHz G5 PowerPC system with 2GB onboard RAM, with 104 dump files totaling 9.54 gigabytes. For comparison, the 104 dump files were loaded into a newly created database (this task took about 9 hours and 10 minutes) on and re-backup using gbak on the same system. The comparative backup using *gbak* took about 26 minutes and restoring the resulting *gbak*-backup-ed file took about 51 minutes. The results for this backup/restore comparison is summarized in Table 1 below for clarity.

	Using gbak utility	This study
Time needed to backup 48 million records in 20 tables	9 hours 10 minutes	22 minutes
Time needed to restore 48 million records in 20 tables	51 minutes	26 minutes

Table 1: Comparison of the length of time needed to backup and restore a test set of 48 million records in 20 tables of a Firebird database using gbak and the proposed method in this study.

4. Discussion

The standard Firebird database backup utility, *gbak*, backs up database into a compressed textual format (XDR), in contrast to MySQL's *mysqldump* utility which backs up database into a plain text format of SQL insert statements. There are advantages and disadvantages in each method. Firebird's method allowed for optimization of the indexes during the backup and restore process while MySQL's method enabled single table backup which tends to be more resilient to data corruption in the backup file. The main disadvantage of Firebird's method is its requirement for an uncorrupted database structure to be backed up successfully while MySQL's method does not allow certain optimizations of the database, such as index balancing, during the backup/restore process. Despite the advantages and disadvantages of each method, there are practical situations where one method might be preferred over another.

This paper presented a simple utility based on Python/kinterbasdb setup for table-level backup and restore in a text-coded format, using *marshal* module in Python Standard Library. The evaluation tests demonstrated that this utility performed with reasonable efficiency of backing up about 2 to 2.2 million records in a minute on a production system which is comparable to that of gbak. The evaluation results illustrated that restoring the database using the proposed utility is significantly slower than that of gbak, this might be use to the serialized use of the restore utility. That is, the dump files are restored one at a time. However, several instances of the restore utility (fb_load.py) could be executed in parallel (restoring different sets of dump files in each instance) and might significantly improve performance, especially on a multi-CPU/multi-core platform with Firebird classic version which allows for multiple instance of Firebird to be concurrently active.

As this utility is based on *marshal* module, Firebird tables containing un-marshallable objects,

such as timestamp or date, cannot be dumped without conversion as shown in the evaluation test where *update_date* attribute (which is date type) had to be casted into 25-character attribute before serialization is possible. Hence, there are some limitations of this proposed method posed by data type mismatch between Firebird data types and marshallable data types. Despite the limitations, it might be possible to circumvent the limitation by type-casting as demonstrated in the evaluation test. Future improvements into the increasing range of serializable data types by *marshal* module may assist in reducing this limitation observed. At the same time, this utility had not been tested with BLOB objects in spite of kinterbasdb's capabilities to handle BLOBs using file-like input/output streams (please refer to kinterbasdb's usage guide for more information). Therefore, we believe that BLOB object handling may be currently possible with modifications and may be included in future versions.

Comparing with gbak, restoring a database using this utility does not result in an optimized set of indexes. However, this could be easily remedied as re-creating table indexes in Firebird is simply a 2-step process of dropping (deleting) a current index (using DROP INDEX SQL statement) and re-creating the same index (using CREATE INDEX SQL statement). These index re-creation could easily be made into an SQL script for routine database maintenance; hence, is not considered as an important feature for this utility.

In summary, we had presented a simple text-formatted backup and loading utility to perform table-by-table backup of a Firebird database for serialize-able or type-cast-able to serialize-able Firebird data types.

5. References

BORRIE, HELEN. (2004) The Firebird Book: A Reference for Database Developers. United States of America, APress.

PALAPARAMBIL, SUDHEER. (2006) October 14, 'Updating Remote DB'. Firebird Support Forum [online]. Available at http://tech.groups.yahoo.com/group/firebird-support/message/80223. Last accessed on November 11, 2006.

PIRIDI, RAMBABU. (2006) November 3, 'Regarding Backup of Firebird Tables'. Firebird Support Forum [online]. Available at http://tech.groups.yahoo.com/group/firebird-support/message/80770. Last accessed on November 11, 2006.

SRINIVASAN, RAJ. (1995) XDR: External Data Representation Standard. Network Working Group. Request for Comments 1832.

VINKENOOG, PAUL. (2005) Firebird's nBackup Tool. Available at http://firebird.sourceforge.net/manual/nbackup.html. Last accessed on January 22, 2007.

Python Events

Title	*Location*	*Dates*
	February	
PyCon 2007	Addison (Dallas), Texas	February 23-25, 2007
URL: http://us.pycon.org/		
	March	
Python course taught by Graham Ellis.	Melksham, England	March 12-14, 2007
URL: http://www.wellho.net/course/ypfull.html		
	April	
RuPy 2007 Ruby and Python conference	Poznan, Poland	April 7-8, 2007
URL: http://rupy.wmid.amu.edu.pl/		
	May	
Python course taught by Graham Ellis.	Melksham, England	May 14-16, 2007
URL: http://www.wellho.net/course/ypfull.html		
Advanced Python course taught by Wesley Chun	San Francisco, CA	May 16-18, 2007
URL: http://roadkill.com/~wesc/cyberweb/pp2dsc.html		
	June	
Python training class taught by Mark Lutz.	Longmont, Colorado	June 11-13, 2007
http://home.earthlink.net/%7Epython-training/longmont-public-classes.htm		

To list your event details here, or to include additional information, please contact editor@pythonpapers.org