OpenVMS Technical Journal V6

Using Python for OpenVMS Web Development

Using Python for OpenVMS Web Development

Author: Jean-Francois Pieronne

A few years ago, a large company whose data-processing architecture is built around OpenVMS were seeking a “Webification” solution for its applications. This company naturally wished to deploy this solution on OpenVMS.

A major criteria criterion was to offer the level of security and audit trail auditing equivalent to that offered by the classic character-cell based applications on OpenVMS using a VT terminal. It was therefore imperative to keep the authentication and security of OpenVMS.

Each user must run his applications in his own process just as in the case of a VT terminal session, only the interface has changed.

A system was needed that was sufficiently light in resource consumption so as to be able to instantiate each user, without an excessive consumption merely supporting such an infrastructure, and providing the required security, identification, and traceability.

Usually application servers like Tomcat or equivalent commercial offerings do not have this level of granularity because of their heavy resource usage.

For reasons of functionality, integration with OpenVMS, and performance, the WASD server package was selected.

Other criteria for the choice of the language for development included:

- Productivity in the creation and maintenance of the programs
- Wealth of libraries, existing tools
- Size of the installed base and the reputation of the language

A scripting language was preferable, due to the much higher productivity of developers in these types of languages than in other more traditional environments.

Finally, Python was selected as the principal language of development because, in conjunction with the WASD web server, it gave the best possibility of successfully deploying the required environment.

The Python Language

Python is a *portable, interpreted, interactive, object-oriented* programming language.

Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high level dynamic data types, and strong dynamic typing. New built-in modules are easily written in C or C++. Python is also usable as an extension language for applications that need a programmable interface.

Python runs on most operating systems and for each has interfaces to many system calls and libraries.

The first public version was released in 1991. In 2001, Guido van Rossum, who is the author of Python, started a foundation named "Python Software Foundation," devoted to advancing open source technology related to the Python programming language.

The Python implementation is copyrighted but freely usable and distributable, even for commercial use.

Since its origin this language has met a growing success, its user-base doubling each year. Many books, articles and Internet sites are regularly devoted to it. Python is currently used by many companies such as Google, Yahoo, Redhat, Microsoft, Nokia, and others.

In the OpenVMS world, several companies use Python intensively; some of them have based their whole Web architecture on the WASD/Python combination.

One company that deploys a software system on hundreds of systems also uses Python intensively, including critical applications.

Here is an example of a script used to replace all the events of a character string in a whole series of HTML files contained in a directory:

```
import glob
for fn in glob.glob('./*.html'):
    res = open(fn).read().replace('old string', 'the new string')
    open(fn, 'w').write(res)
```

For more information and other demonstrations to refer to the site http://www.python.org.

Porting Python

A first port of Python 1.5.2 had been carried out, but for technical reasons and to ensure easier development in the future, it was decided to set out again from scratch.

Currently, this port is the result of collaborative work among several people, with the occasional participation of HP employees.

The current version of Python for OpenVMS is 2.3.5. New kits are periodically released, including bug fixes, or updates of the libraries contained in this port.

Since version 2.3.5, as the kits are now versioned, it is possible to have several versions of Python installed on the same system.

The Challenge

We had to ensure a port that is perennial and evolutionary, that facilitates the port of future versions, and is not a one-shot port.

This includes submission of the patches carried out specifically for OpenVMS for inclusion into the main stream Python codebase. This has been partially accomplished. However, there still exist some patches that are not included. Indeed, this is sometimes difficult. For example, it was necessary to explain how a file system like RMS works, because most people think that a file is limited to a flow of bytes.

We also had to ntegrate specific OpenVMS features, for example:

- Allow the calling of system services or various runtime library routines (SYSxxx, LIBxxx etc)
- Offer access to RMS files and not be limited to sequential files with organisation type of STREAMLF
- Utilize environments specific to OpenVMS such SGBD Oracle/Rdb or the WASD web server
- Allow the addition of modules or the evolution of existing modules without having to rebuild the whole interpreter.

- Allow the simplest possible port, under OpenVMS, of Python applications initially developed for other OS. For example, it was decided that for Python OpenVMS is seen as a posix system and that the routines for access to the file-system followed posix naming rules and not those of OpenVMS naming rules.
- Provide a form of installation integrated into OpenVMS.

The Choices We Made

The Python interpreter is built as a shareable image. The Python program itself is only one small program of 24 lines.

The extensions for OpenVMS are also built in the form of shareable images. Therefore, it is simple to upgrade a module or add new ones.

Moreover, as these libraries are dynamically activated, it is possible to deploy Python with all of the modules provided, on systems that do not have the specific environments installed (for example Oracle/Rdb or the Web server WASD).

When the modules use libraries that can also be used outside of Python, this functionality is provided in the form of separate kits.

Here is the list of the libraries for which there is a PCSI kit and a Python interface:

- Zlib
- LibBZ2
- LibJPEG
- LibPNG
- Freetype
- LibImaging
- LibGD
- GDChart
- LibXML2 Libxslt/Libexslt
- OPENSSL
- Swish-E

The OpenVMS Python kit and all the separate libraries are distributed in the form of PCSI kits and are therefore easily installable and upgradable.

It was also decided that the minimum supported version of OpenVMS would be V7.3 with the latest version of the CRTL. (There is a partial version for OpenVMS V7.2, but it is not maintained.)

It is strongly advisable to install Python on a ODS-5 disk. However, in the absence of a physical disk of this type, it is possible to use a virtual disk in a container file with LDdriver, the latter being included with OpenVMS starting from the V7.3-1. LDdriver for older versions of OpenVMS is available on the Freeware CD. (Editor's Note: Look for the article about the LDdriver in this issue of the OpenVMS Technical Journal.)

Some Examples of Integration with OpenVMS

Example 1

Display users whose accounts are not disusered and who have not logged on for 31 days or more:


```
import vms.user, vms.starlet, vms.uaidef
def fcmp(u1, u2):
    return cmp(u2.lastlogin_i, u1.lastlogin_i)
users = vms.user.all_users()
users = users.values()
# descending sort on last login interactive
users.sort(fcmp)
s,delta = vms.starlet.bintim('31 0:0:0.00')
s,curtim = vms.starlet.asctim()
s,minlogin = vms.starlet.bintim(curtim)
minlogin += delta
for user in users:
    if (not (user.flags & vms.uaidef.UAI_M_DISACNT) and
        0 < user.lastlogin_i < minlogin):
        print "%-33s %s" % (user.username,
                            vms.starlet.asctim(user.lastlogin_i)[1])
```

This gives the following result:

```
USER1                                8-APR-2005 13:38:43.51
FIELD                                8-APR-2005 13:16:48.21
DEMO                                27-SEP-2002 12:49:57.24
```

Example 2

Sample code to fetch information from queues, display all queue names and descriptions and for each queue all jobs name and id:

```
from vms.rtl.lib import getqui
from vms.quidef import *
from vms.jbcmsgdef import JBC__NOMOREQUE, JBC__NOSUCHJOB,\
                          JBC__NOMOREJOB
getqui(QUI__CANCEL_OPERATION)
while True:
    queue_name = ''
    try:
        s, v, queue_name = getqui(QUI__DISPLAY_QUEUE, QUI__QUEUE_NAME,
                                  None, '*')
    except VMSError, e:
        if e.errno  == JBC__NOMOREQUE:
            break
        else:
            raise
    s, v, queue_desc = getqui(QUI__DISPLAY_QUEUE,QUI__QUEUE_DESCRIPTION,
                              None, '*', QUI_M_SEARCH_FREEZE_CONTEXT)
    print 'Queue:', queue_name, '<', queue_desc, '>'
    while True:
        try:
            s, v, js = getqui(QUI__DISPLAY_JOB, QUI__JOB_STATUS,-1,None,
                              QUI_M_SEARCH_ALL_JOBS)
        except VMSError, e:
            if e.errno in (JBC__NOMOREJOB, JBC__NOSUCHJOB):
                break
            else:
                raise
        s, v, jn = getqui(QUI__DISPLAY_JOB, QUI__JOB_NAME, -1, None,
                          QUI_M_SEARCH_ALL_JOBS |
                                  QUI_M_SEARCH_FREEZE_CONTEXT)
        s, v, en = getqui(QUI__DISPLAY_JOB, QUI__ENTRY_NUMBER, -1, None,
                          QUI_M_SEARCH_ALL_JOBS |
```

```
                                        QUI_M_SEARCH_FREEZE_CONTEXT)
        print '    Job: %s (%s)' % (jn, en)
```

This gives the following result:

```
Queue: ASSP_QUEUE < queue for Anti-Spam SMTP Proxy Server >
    Job: LOGIN_ASSP (6)
Queue: LASER1 <  >
Queue: SCHEDULER <  >
    Job: IPCHECK (562)
    Job: CHECKAUDIT (292)
    Job: update_spamdb (530)
Queue: SETI$BATCH <  >
Queue: SYS$BATCH_NODE1 < Queue batch NODE1 >
Queue: SYS$BATCH_GENERIC < Queue batch generic >
Queue: TCPIP$SMTP_NODE1_00 <  >
```

Example 3

A small Rdb example which displays the user tables and views from the famous "mf_personnel" demonstration database. An iterator is used to read the lines.

```
import rdb
attach = rdb.statement("attach 'filename mf_personnel'")
commit = rdb.statement("commit work")
readonly = rdb.statement("set transaction read only")
curs = rdb.statement("""select rdb$relation_name from rdb$relations
where rdb$system_flag = ? order by rdb$relation_name""")
attach.execute()
print "users relations name:"
readonly.execute()
curs.execute(0)
for line in curs:
    print line[0]
commit.execute()
```

Which gives the following results:

```
users relations name:
CANDIDATES
COLLEGES
CURRENT_INFO
CURRENT_JOB
CURRENT_SALARY
DEGREES
DEPARTMENTS
EMPLOYEES
JOBS
JOB_HISTORY
RESUMES
SALARY_HISTORY
WORK_STATUS
```

For more information and other examples, refer to the site Python for OpenVMS.

Applications, Libraries, and Tools

It is generally easy to port a Python application to OpenVMS provided that it doesn't use platform specific libraries. For example, the porting of the application server Webware which represents around 40,000 lines of code required only the modification of a single line in the installation script. A PCSI installation kit for Webware will be shortly available.

Many other tools & libraries are installed and use the standard installation procedure without any modification for example the the Cheetah templating system which comprises around 12,000 lines of Python code or the PyChecker tool which is a Python source code checking tool and comprises approximately 5,000 lines of Python code.

A Web T4 viewer has been developed in a few days and comprises about only 900 lines of Python code. A demonstration is available from http://vmspython.dyndns.org/.

For More Information

Some interesting reading may be:

- The article from John K. Ousterhout creator of the TCL language: Scripting: Higher Level Programming for the 21st Century
- The article from Ill Venners: Use the Best Tool for the Job
- The comparative study written by Lutz Prechelt: An Empirical Comparaison of C, C++, Java, Perl, Python, Rexx, and Tcl.
- Artima.com's six-part interview with Python creator Guido van Rossum: http://www.artima.com/intv/guido.html
- The following presentations from Stefen Ferg:

  - The Business Case for Agile Languages
  - Python:a Powerful, Easy-to-Use, Open-Source Scripting Language
  - Python & Java: a Side-by-Side Comparison
  - More material can be accessed from http://www.ferg.org/python_presentations/index.html.

For more information about T4, read the following article by Steve Lieman a previous OpenVMS Technical Journal:

TimeLine-Driven Collaboration with "T4 & Friends": A Time-saving Approach to OpenVMS Performance