Accessing Zen Edge v13 from Python on Raspbian Using the Btrieve 2 Interface

A White Paper From

For more information, see our web site at
http://www.goldstarsoftware.com

Accessing Zen Edge v13 from Python on Raspbian Using the Btrieve 2 Interface

Last Updated: June 2019

The Actian Zen database engine (formerly known as Actian PSQL) supports a wide variety of application programming interfaces (APIs) to access the data. Some of these interfaces leverage the power of SQL to access your data, while others use a lower-level interface, commonly known as the Btrieve API to provide the needed performance and flexibility.

With the introduction of the Zen Edge environment on hardware with limited capabilities, we fully expect the new Btrieve 2 API to become a favorite for developers. This new interface allows developers to create code using scripting languages (like Python, Perl, etc.), as well as other higher-order languages like C#. These languages will simplify the development of simple data collection and analysis applications, especially in a web-based or IoT (Internet of Things) environment. More importantly, the Btrieve 2 API offers developers the ability to maintain the high performance and extreme flexibility they have come to know and expect from the Zen/PSQL database engine, all in a tiny footprint.

This paper is broken up into each of the critical steps you need to follow in order to get Python working on a Raspberry Pi running Raspbian Linux. It is complicated because it is bringing together many different components, many of them open source, so that they all work together. The process was first documented by Actian's own Linda Anderson in a blog post, but we wanted to give you a little bit more information to help jumpstart the process. In addition, many developers today are familiar only with the Microsoft market (i.e. DOS and Windows), so Linux can often be a great mystery. This is the main reason why we created this document: to provide specific commands and explain each of the steps in detail. This document uses "**$**" to indicate commands that are to be executed at the command prompt. You will only type the characters that appear AFTER the "**$**" symbol.

Note that while the code in Appendix A contains an extremely simple example of various operations for you, it does NOT explain the ins and outs of the Btrieve 2 API itself. For more detailed information on the Btrieve 2 API, consult the Actian web site at http://docs.actian.com/psql/psqlv13/btrieve2api/wwhelp/wwhimpl/common/html/wwhelp.htm#href=btrieve2api.htm&single=true. Note, though, that this documentation is specific to the C++ environment. Since Python can determine the length of parameters from the parameters themselves, the length parameters are not passed. If you have specific questions about parameters needed for a call, check out the **btrievePython.swig** file.

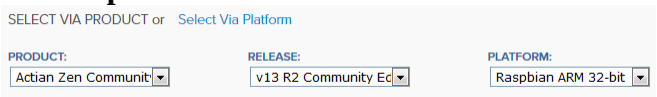
So, get out your Pi with a fresh Raspbian installation, and let's get this working!

Downloading and Installing the Zen Edge Engine

The first piece you need (after installing Raspbian, of course), is the Zen Edge Engine. The following steps will get the engine installed onto the Pi:

1) Go to https://esd.actian.com/product/Zen_PSQL in a web browser.

2) In the boxes provided, select **Actian Zen Community Edition**, **v13 R2 Community Edition,** and **Raspbian ARM 32-bit**.



3) Scroll down and open up the link for **Zen Edge v13 R2 Community Edition**, then click on the HTTP Download button for the Zen database.

4) Open up a shell window to your Raspbian system (either locally or remote) and log in as the default `pi` user.
5) Copy the file from the `Downloads` folder to the `/usr/local` folder. *Note that your filename may be different as this download is updated, so adjust the commands accordingly. One handy feature of Linux command prompt is "tab completion", so you can just start typing and press the <Tab> key to complete the filename.*

```
$ cd Downloads
$ sudo cp Zen-IoT-Community-linux-13.31-006.000.armhf.tar.gz /usr/local
```

6) Install the Zen IoT Server Engine using these commands:

```
$ cd /usr/local
$ sudo tar -zxf Zen-IoT-Community-linux-13.31-006.000.armhf.tar.gz
$ cd psql/etc
$ sudo ./preinstall.sh
$ sudo ./postinstall.sh
```

Creating a Development Account and Working Folder

While the default `pi` user is good for many things, you will probably want to use a separate account for your development efforts to avoid cluttering the `/home/pi` folder structure. Create a user (we'll use `bill` in this example) that is in both the `sudo` and `pvsw` groups:

```
$ sudo adduser bill
$ sudo adduser bill sudo
$ sudo adduser bill pvsw
```

Now, log out of the `pi` user and log back into your newly-created user account.

To store all of your development files, we're going to need a storage location to work from. This can be whatever and wherever you want it to be. In this example, we are going to create a folder inside the user's home folder called **`MyPrograms`**, and we will refer to that folder from here on out. This command will create this folder for the user `bill` (and will work from any login):

```
$ cd
$ mkdir MyPrograms
```

Of course, if you use a different folder name, just be sure to change all references to this folder in each of the sample command lines that follow.

The final step here is to configure this user's environment, so that the database utilities and library files can be found when they are needed. We are going to simply concatenate the `.bashrc` file from the `psql` user (which was created by the Zen installer) to our current file:

```
$ cd
$ cat /home/psql/.bashrc >>.bashrc
```

You should log out and log back in one more time, so that your environment is properly configured.

Checking Your Python Version

It is important to know which version of Python is installed on your Raspbian environment, as most systems come with at least some version pre-installed for you.

1) Open up a shell window to your Raspbian system (either locally or remote) and log in.
2) Check your current version of Python with this command:
   ```
   $ python --version
   or
   $ python3 --version
   ```
3) If you already have a usable version, then you can jump right to the SWIG installation. Otherwise, the next section can help you to get the latest Python version installed.

Downloading and Installing Python for Raspbian (Optional)

If you are going to do development work, the version of Python included with Raspbian may be sufficient for you, so just proceed to the next section. If you would rather use the latest and greatest version of Python, you can follow these steps to download and install the current release. Note, though, that this process can easily take 3 hours or more, so you might want to plan on letting this run this overnight!

1) Open up a shell window to your Raspbian system (either locally or remote) and log in.
2) Although Raspbian seems to lag behind other Linux distros, it is worth at least trying to install using apt-get to save some time, so try these commands first.
   ```
   $ sudo apt-get update
   $ sudo apt-get install python3.6
   ```
3) If the above worked, then you can stop here and thank the Linux gods for your good fortune!
4) If not, then you're going to need to build your own version of Python from the source code. Start by installing the required build tools with these commands:
   ```
   $ sudo apt-get update
   $ sudo apt-get install build-essential tk-dev libncurses5-dev libncursesw5-dev
   libreadline6-dev libdb5.3-dev libgdbm-dev libsqlite3-dev libssl-dev libbz2-dev
   libexpat1-dev liblzma-dev zlib1g-dev
   ```
   Note that the second INSTALL line has been broken up onto multiple lines, just to make it easier to read in this format. You should copy and paste this entire line (without the **$**). On my test system, this step took 2-3 minutes to complete.
5) Download and build the latest Python environment. At the time of this writing, the current version is 3.6.4. The following commands will handle this for you, but be sure to check the latest version at www.Python.org and update the version number accordingly. Note that while some web sites have indicated that these steps can take "as long as 15 minutes to run", the limited speed of the Pi means that this will take MUCH longer – as much as 3 hours or more! The `-enable-optimizations` flag is part of why this takes so long, but it does provide a 10% performance gain at runtime, so it should be worth the extra effort. If you are in a hurry, though, then you can leave this switch off for a faster build time.
   ```
   $ wget https://www.python.org/ftp/python/3.6.4/Python-3.6.4.tar.xz
   $ tar xf Python-3.6.4.tar.xz
   ```

```
$ cd Python-3.6.4
$ ./configure --enable-optimizations
$ make
$ sudo make altinstall
```

6) If the build was successful, you will now have a complete build of Python 3.6.4 in the `/usr/local/bin` folder. You can now remove the Python source code and the build components to reclaim storage space. The following commands will clear out the extra gunk:

```
$ sudo rm -r Python-3.6.4
$ rm Python-3.6.4.tgz
$ sudo apt-get --purge remove build-essential tk-dev
$ sudo apt-get --purge remove libncurses5-dev libncursesw5-dev libreadline6-dev
$ sudo apt-get --purge remove libdb5.3-dev libgdbm-dev libsqlite3-dev libssl-dev
$ sudo apt-get --purge remove libbz2-dev libexpat1-dev liblzma-dev zlib1g-dev
$ sudo apt-get autoremove
$ sudo apt-get clean
```

7) Finally, if you want to use the latest Python exclusively on your system, you will need to make a few more changes to force that version to work when you call "python3". These statements will set up a symbolic link to the new Python executable code called "python3", ensure that the correct location is in your path (which may already be done), and then reset the command hash:

```
$ cd /usr/local/bin
$ sudo ln -s /usr/local/bin/python3.6 python3
$ PATH=/usr/local/bin:$PATH            (Only do this if not already in $PATH.)
$ hash -r
```

For additional ways to set up an alternative Python version for one user or for the entire system, check out https://linuxconfig.org/how-to-change-from-default-to-alternative-python-version-on-debian-linux.

Downloading and Installing SWIG

The Simplified Wrapper and Interface Generator (SWIG) components must be installed next:

1) Open up a shell window to your Raspbian system (either locally or remote) and log in.
2) Execute the following two commands:

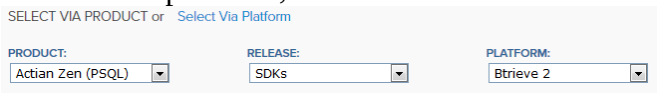
```
$ sudo apt-get update
$ sudo apt-get install swig
```

3) There are a few other related packages (such as examples and documentation) that are not installed by default. If you do want them installed, you can do that with another `apt-get` command. Check the messages reported from this process for the names of those components.

Downloading and Installing the Btrieve 2 SDK

The next piece you need is the SDK download for Btrieve 2. If you have the SDK already exploded on a Windows system, then you can simply copy the files from there. If not, the following steps will explode the files directly on the Pi:

1) Go to https://esd.actian.com/product/Zen_PSQL in a web browser.
2) In the boxes provided, select **SDKs** and **Btrieve 2**.



3) Scroll down and open up the link for **Btrieve 2**, then click on the HTTP Download button for the Btrieve 2 Linux SDK for PSQL 13.

4) Open up a shell window to your Raspbian system under your account.
5) Change to your Downloads folder and unzip the package. *Note that your filename and subsequent directory may be different as this download is updated, so adjust accordingly.*
```
$ cd Downloads
$ tar xf PSQL-SDK-Btrieve2API-13.30.034.000-Linux-noarch.tar.gz
```
6) This will explode the needed Btrieve 2 API components. Copy the following two files from the "swig" folder to your MyPrograms folder:
```
$ cd PSQL-SDK-Btrieve2API-13.30.034.000-Linux-noarch
$ cp swig/btrievePython.swig ~/MyPrograms
$ cp swig/btrieveSwig.swig ~/MyPrograms
```
7) Finally, copy the following two files from the "include" folder to your working folder:
```
$ cp include/btrieveC.h ~/MyPrograms
$ cp include/btrieveCpp.h ~/MyPrograms
```

Building the SWIG Wrapper

Building the SWIG wrapper is the last setup step. Note that as the various components change, such as Python, SWIG, and the Btrieve 2 API, you may need to re-run this process to build a new SWIG wrapper from time to time.

1) Open a command prompt and change to your MyPrograms folder:
```
$ cd ~/MyPrograms
```
2) Build the SWIG Wrapper with this command:
```
$ sudo swig -c++ -D__linux__ -python btrievePython.swig
```
If this process is successful, you will have two new files in your current folder, namely **btrievePython.py** and **btrievePython_wrap.cxx**.
3) Use the command "vi setup.py" to create a new file and paste in the following text:
```
#!/usr/bin/env python
from distutils.core import setup, Extension
btrievePython_module = Extension('_btrievePython',
        sources=['btrievePython_wrap.cxx'],
        library_dirs=['/usr/local/psql/lib/'],
        runtime_library_dirs=['/usr/local/psql/lib'],
        libraries=['btrieveCpp'] )
setup (name='btrievePython',
        version='1.0',
        author='Actian',
        description="""Compile Btrieve 2 Python module""",
        ext_modules=[btrievePython_module],
        py_modules=["btrievePython"], )
```
4) Run this command to build the btrievePython module against Python3:
```
$ sudo python3 setup.py build_ext --inplace
```
You may see some warnings which can be ignored. When it is complete (about 30 seconds or so), you should now see the file **_btrievePython.cpython-36m-arm-linux-gnueabihf.so** in the working folder. Note that your file and folder name will vary depending on the exact version of Python and the CPU you are running. For example, if you are running Python 3.5, then you will get the file **_btrievePython.cpython-35m-arm-linux-gnueabihf.so** instead.

5) Rename the newly-created file to _btrievePython.so with this command line. (You can copy & paste this command line for best results, but note that the specific ARM CPU and Python version numbers are embedded in the file name, so make any changes as needed.)

```
$ mv _btrievePython.cpython-36m-arm-linux-gnueabihf.so _btrievePython.so
  or
$ mv _btrievePython.cpython-35m-arm-linux-gnueabihf.so _btrievePython.so
```

Note that the files _btrievePython.so (generated in step 5) and btrievePython.py (generated in step 2) must be accessible by your Python application. Once you have followed these steps for a given CPU and Python version, you can simply copy them around as needed.

Creating your Python Application

Once you've jumped through the above hoops, you can now import the newly-created module in order to access the Btrieve 2 API from with your code:

```
import sys
sys.path.append('/usr/local/psql/lib')
import btrievePython
```

The sample code in Appendix A shows examples of several key operations, including:

- Creating a New File
- Opening an Existing File
- Creating a New Index on an Existing File
- Inserting Records
- Reading All Records in Key Order
- Performing a Record Lookup

If you want to try this sample application as is, create a new text file called **test_btr2.py**, then copy and paste in the source code from Appendix A into the file and save it.

Running Your Python Application

Running the application fairly easy. Simply launch Python and pass in the script name!

```
$ python3 test_btr2.py
```

From here, you're only limited by your imagination!

Finding More Help

If you have other problems getting this to work, we urge you to contact Actian directly through their web forums at https://communities.actian.com/s/ for more help. If you need some additional hand-holding, Goldstar Software may be able to assist you as well. You can contact us at 1-708-647-7665 or via the web at http://www.goldstarsoftware.com.

Appendix A: Sample Application

The following application makes use of the Btrieve 2 API and can be used as instructional and sample code.

```
import os
import sys
import struct
sys.path.append('/usr/local/psql/lib')
import btrievePython as btrv

btrieveFileName = "/usr/local/psql/data/DEMODATA/Test_Table.mkd"
recordFormat = "<iB32sBBBH"
recordLength = 42
keyFormat = "<i"
key1Format = "B32s"

# Create a session:
btrieveClient = btrv.BtrieveClient(0x4232, 0)  #B2
# Specify FileAttributes for the new file:
btrieveFileAttributes = btrv.BtrieveFileAttributes()
rc = btrieveFileAttributes.SetFixedRecordLength(recordLength)
# Specify Key 0 as an autoinc:
btrieveKeySegment = btrv.BtrieveKeySegment()
rc = btrieveKeySegment.SetField(0, 4, btrv.Btrieve.DATA_TYPE_AUTOINCREMENT)
btrieveIndexAttributes = btrv.BtrieveIndexAttributes()
rc = btrieveIndexAttributes.AddKeySegment(btrieveKeySegment)
rc = btrieveIndexAttributes.SetDuplicateMode(False)
rc = btrieveIndexAttributes.SetModifiable(True)

# Create the file:
rc = btrieveClient.FileCreate(btrieveFileAttributes, btrieveIndexAttributes,
       btrieveFileName, btrv.Btrieve.CREATE_MODE_OVERWRITE)
if (rc == btrv.Btrieve.STATUS_CODE_NO_ERROR):
    print('File ', btrieveFileName, ' created successfully!')
else:
    print('File ', btrieveFileName, ' not created; ', rc, ': ',  btrv.Btrieve.StatusCodeToString(rc))

# Allocate a file object:
btrieveFile = btrv.BtrieveFile()
# Open the file:
rc = btrieveClient.FileOpen(btrieveFile, btrieveFileName, None, btrv.Btrieve.OPEN_MODE_NORMAL)
if (rc == btrv.Btrieve.STATUS_CODE_NO_ERROR):
    print('File open successful!')
else:
    print('File open failed - status: ', rc,  ': ',  btrv.Btrieve.StatusCodeToString(rc))

# Create Key 1 as a String with Null Indicator Byte:
btrieveKey1aSegment = btrv.BtrieveKeySegment()
rc = btrieveKey1aSegment.SetField(4, 1, btrv.Btrieve.DATA_TYPE_NULL_INDICATOR_SEGMENT)
rc = btrieveKey1aSegment.SetDescendingSortOrder(True)
btrieveKey1bSegment = btrv.BtrieveKeySegment()
rc = btrieveKey1bSegment.SetField(5, 32, btrv.Btrieve.DATA_TYPE_CHAR)
btrieveIndex1Attributes = btrv.BtrieveIndexAttributes()
rc = btrieveIndex1Attributes.AddKeySegment(btrieveKey1aSegment)
rc = btrieveIndex1Attributes.AddKeySegment(btrieveKey1bSegment)
rc = btrieveIndex1Attributes.SetDuplicateMode(btrv.Btrieve.DUPLICATE_MODE_ALLOWED_NONREPEATING)
rc = btrieveIndex1Attributes.SetModifiable(True)
rc = btrieveFile.IndexCreate(btrieveIndex1Attributes)
if (rc == btrv.Btrieve.STATUS_CODE_NO_ERROR):
    print('Index 1 created successfully!')
else:
    print('Index 1 not created; error: ', rc, ': ',  btrv.Btrieve.StatusCodeToString(rc))

# Insert records:
iinserting = True
print('\n')
while iinserting:
    new_name =  input('Insert name (Q to quit): ' )
```

```
    if new_name.lower() == 'q':
        iinserting = False
    else:
        record = struct.pack(recordFormat, 0, 0, new_name.ljust(32).encode('UTF-8'), 0, 22, 1, 2018)
        rc = btrieveFile.RecordCreate(record)
        if (rc == btrv.Btrieve.STATUS_CODE_NO_ERROR):
            print('   Insert successful!')
        else:
            print('   Insert failed - status: ', rc,  ': ',  btrv.Btrieve.StatusCodeToString(rc))

# Get Record count:
btrieveFileInfo = btrv.BtrieveFileInformation()
rc = btrv.BtrieveFile.GetInformation(btrieveFile, btrieveFileInfo)
print('\nTotal Records inserted =', btrieveFileInfo.GetRecordCount())

# Display all records in sorted order
print('\nHere is a list of the names in alphabetical order:')
readLength = btrieveFile.RecordRetrieveFirst(1, record, 0)
while (readLength > 0):
    recordUnpacked = struct.unpack(recordFormat, record)
    print('   ID:', recordUnpacked[0], ' Name:', recordUnpacked[2].decode())
    readLength = btrieveFile.RecordRetrieveNext(record, 0)

# Look up record by name
ireading = True
while ireading:
    find_name =  input('\nFind name (Q to quit): ' )
    if find_name.lower() == 'q':
        ireading = False
    else:
        key1Value = struct.pack(key1Format, 0, find_name.ljust(32).encode('UTF-8'))
        readLength = btrieveFile.RecordRetrieve(btrv.Btrieve.COMPARISON_EQUAL, 1, key1Value, record)
        if (readLength > 0) :
            recordUnpacked = struct.unpack(recordFormat, record)
            print('   Matching record found: ID:', recordUnpacked[0], ' Name:', recordUnpacked[2].decode())
        else:
            print('   No record found matching "'+find_name+'"')
            status = btrieveFile.GetLastStatusCode()
            if (status != 4):
                print('   Read error: ', status, ': ', btrv.Btrieve.StatusCodeToString(status))

# Close the file:
rc = btrieveClient.FileClose(btrieveFile)
if (rc == btrv.Btrieve.STATUS_CODE_NO_ERROR):
    print('File closed successfully!')
else:
    print('File close failed - status: ', rc, ': ', btrv.Btrieve.StatusCodeToString(status))
```