Accessing Zen v15 from Python on Windows Using the Btrieve 2 Interface

A White Paper From

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

Accessing Zen v15 from Python on Windows Using the Btrieve 2 Interface

Last Updated: September 2022

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 IoT 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 simplify the development of simple data collection and analysis applications, especially in a web-based or IoT (Internet of Things) environment running on Zen Core (for iOS or Android) or Zen Edge. 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 with Zen v15 on Windows. It is complicated because it brings 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 you jumpstart the process, so we created this document to smooth out the process. 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.

Because of this complexity, we recommend that you create a development folder so that you can keep everything in one place on your development computer. Our example here will use the folder "C:\Develop" as the base folder, and everything else will be installed under that location. Of course, if you use an alternative location, just be sure to change "C:\Develop" to your own root path in all of the sample code.

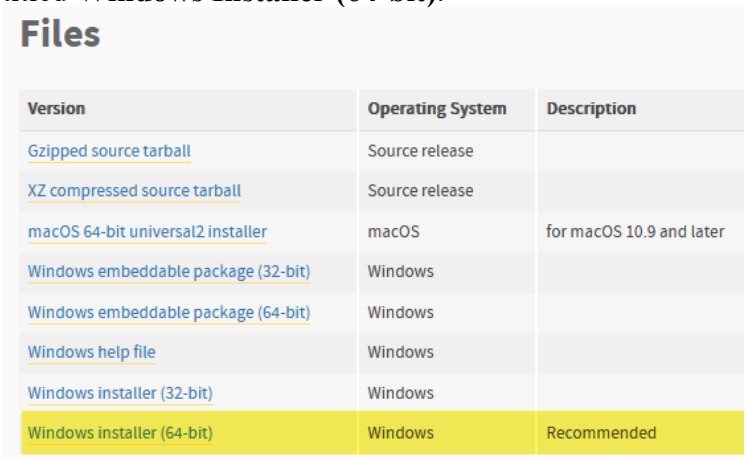
Note that while the code in Appendix A contains some examples of various operations for you, it does NOT explain the ins and outs of the Btrieve 2 API itself. For more detailed information on the Btrieve 2 API, please consult the Actian web site at https://docs.actian.com/psql/btrieve2v13/html/. Note, though, that this documentation is specific to the C/C++ environment. Since Python can determine the length of parameters from the parameters themselves, the length parameters do not need to be passed. If you have specific questions about parameters needed for a call, check out the **btrievePython.swig** file.

Downloading and Installing Python for Windows

If you don't have Python set up yet, then this becomes the first obvious step.

1) In your web browser, go to https://www.python.org/downloads/windows/ and select the version of Python you want to download. At the time of this writing, the current version is 3.10.7.

2) From the second page, click on the link to download the Executable Installer for the platform on which you want to install. For a typical x64 Windows environment, we recommend the link titled **Windows Installer (64-bit)**.



Files

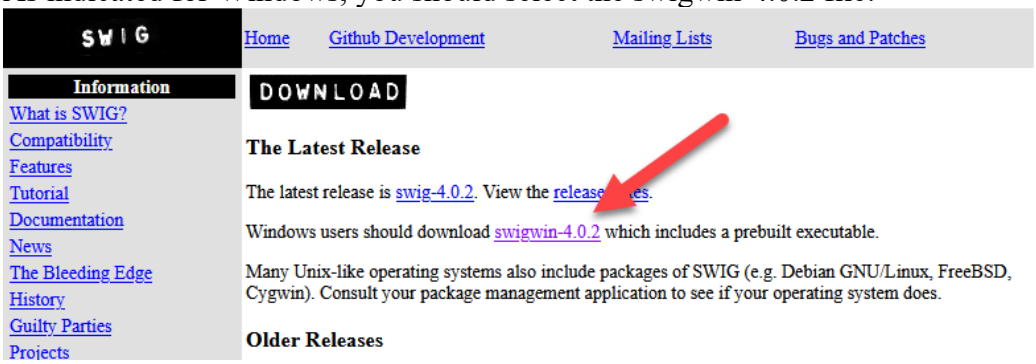
Version	Operating System	Description
Gzipped source tarball	Source release	
XZ compressed source tarball	Source release	
macOS 64-bit universal2 installer	macOS	for macOS 10.9 and later
Windows embeddable package (32-bit)	Windows	
Windows embeddable package (64-bit)	Windows	
Windows help file	Windows	
Windows installer (32-bit)	Windows	
Windows installer (64-bit)	Windows	Recommended

3) Follow the instructions for the default installation and set the installation folder to C:\Develop. When you are done, you will have Python installed in the C:\Develop\Python310 folder and ready to go.
4) Create a folder called **MyPrograms** underneath the Python folder – this will be where we will do all of our development efforts.

Downloading and Installing SWIG

The SWIG components are the second piece:

1) In your web browser, go to http://www.swig.org/download.html and select the version of SWIG you want to download. At the time of this writing, the current version is 4.0.2.
2) As indicated for Windows, you should select the swigwin-4.0.2 file:

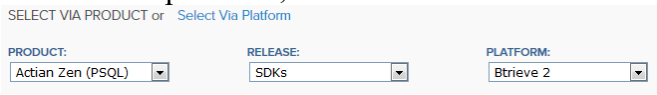


3) There is no installer for SWIGWIN. Instead, simply unzip the entire file into your MyPrograms folder, creating **C:\Develop\Python36\MyPrograms\swigwin-4.0.2** in the process.

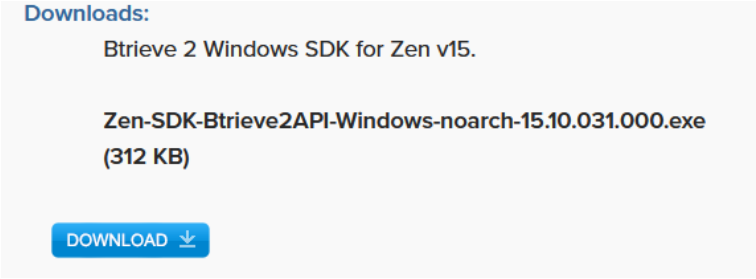
Downloading and Installing the Btrieve 2 SDK

The next piece you need is the SDK download for Btrieve 2:

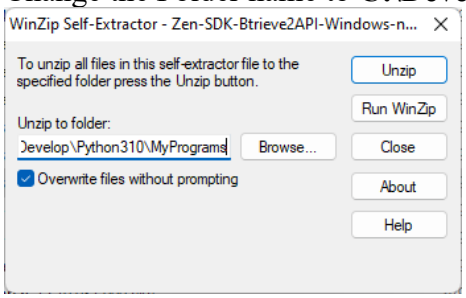
1) Go to https://esd.actian.com/product/Zen_PSQL in a web browser.
2) In the boxes provided, select **SDKs** and **Btrieve 2**.



3) Scroll down and open up the link for **Btrieve 2**, then click on the **Download** button for the Btrieve 2 Windows SDK for Zen v15.



4) After it downloads, double-click the EXE file to launch the installer.
5) Change the Folder name to **C:\Develop\Python310\MyPrograms** and click Unzip:



6) You will now have the Btrieve 2 API components in a folder on your workstation inside the MyPrograms folder.
7) Copy these two files from the "swig" subfolder to your MyPrograms folder:
   `btrievePython.swig`
   `btrieveSwig.swig`
8) Copy these two files from the "include" subfolder to your MyPrograms folder:
   `btrieveC.h`
   `btrieveCpp.h`
9) If you are working on a 64-bit system, copy the following files from the "win64\x86_64" folder to your MyPrograms folder. If you are on a 32-bit OS, copy the files from the "win32\x86" folder instead:
   `btrieveC.lib`
   `btrieveCPP.lib`

Building the SWIG Wrapper

Building the SWIG wrapper is the most complicated part of this entire process. This process requires that you have a development environment on your computer, and this documentation is assuming Visual Studio. However, at the time of this writing, the following compilers were supported:

- --compiler=bcpp Borland C++ Compiler
- --compiler=cygwin Cygwin port of GNU C Compiler for Win32
- --compiler=mingw32 Mingw32 port of GNU C Compiler for Win32
- --compiler=msvc Microsoft Visual C++
- --compiler=unix standard UNIX-style compiler

If you don't have any of these installed, you may be able to find a Community Edition of Visual Studio from Microsoft's web site. Alternatively, ask a developer friend to help with this step!

1) Open a command prompt and change to your MyPrograms folder:
   `CD C:\Develop\Python36\MyPrograms`

2) Build the SWIG Wrapper with this command:
```
swigwin-4.0.2\swig -c++ -D_WIN64 -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**. Note that if you are building SWIG for a different environment (such as Win32 or MacOS), you might need a different –D switch, such as -D_WIN32 or -D__APPLE__.

3) Create a text file called **setup.py** and paste in the following text:
```
from distutils.core import setup, Extension
btrievePython_module = Extension('_btrievePython',
      sources=['btrievePython_wrap.cxx'],
      library_dirs=['c:\Develop\Python310\MyPrograms\ '],
      libraries=['btrieveCpp'],
      )
setup (name='btrievePython',
      version='1.0',
      author='Actian',
      description="""Compile Btrieve 2 Python module""",
      ext_modules=[btrievePython_module],
      py_modules=["btrievePython"],
      )
```
Be sure to change **library_dirs** if you are using a different location! Note that the trailing backslash has a space after it. If you omit this space, then the backslash will escape the single quote, and the next command will fail.

4) Run this command to build the btrievePython module:
```
python setup.py build_ext --plat-name="win-amd64"
```
This will create the compiled Python file **_btrievePython.cp310-win_amd64.pyd** in the build\lib.win-amd64-cpython-310\ folder. If you have problems with this step, see below.

5) Copy the newly-created file into the MyPrograms folder and rename it to _btrievePython.pyd:
```
copy build\ lib.win-amd64-cpython-310\_btrievePython.cp310-win_amd64.pyd .
REN _btrievePython.cp310-win_amd64.pyd _btrievePython.pyd
```

6) (Optional) You can copy the pyd file to the C:\Develop\Python310\DLLs folder as well. This should be done if you will be working within many different folders on the same workstation and you don't want to copy this file around to each of them.

Note regarding step 4 above: The BUILD_EXT script is supposed to be smart enough to detect your compiler. It worked fine with my Visual Studio 2019 installation, but a previous install with VS2013 did not, and Step 4 only returned the minimal error "Unable to find vcvarsall.bat". After much digging through the script files, I eventually found that I could fix this by setting a new environment variable:
```
SET VS140COMNTOOLS=%VS120COMNTOOLS%
```
This allowed the build_ext script to locate the v12 compiler in the v14 location.

In another environment, a partial installation of the Visual Studio Build Tools was available, and the error "**error: Microsoft Visual C++ 14.0 or great is required. Get it with "Microsoft C++ Build Tools": https://visualstudio.microsoft.com/visual-cpp-build-tools/**" was returned. If you see this, re-run the Visual Studio Build Tools installation and verify that the options for "C++/CLI Support" and "VC++ Toolset for Desktop" are selected

When it runs properly, you should see the following output text (or something similar):

```
C:\Develop\Python310\MyPrograms>python setup.py build_ext --plat-name="win-amd64"
running build_ext
building '_btrievePython' extension
creating build
creating build\temp.win-amd64-cpython-310
creating build\temp.win-amd64-cpython-310\Release
"C:\Program Files (x86)\Microsoft Visual
Studio\2019\Professional\VC\Tools\MSVC\14.29.30133\bin\HostX86\x64\cl.exe" /c /nologo /O2 /W3 /GL
/DNDEBUG /MD -IC:\Develop\Python310\include -IC:\Develop\Python310\Include "-IC:\Program Files
(x86)\Microsoft Visual Studio\2019\Professional\VC\Tools\MSVC\14.29.30133\ATLMFC\include" "-IC:\Program
Files (x86)\Microsoft Visual Studio\2019\Professional\VC\Tools\MSVC\14.29.30133\include" "-IC:\Program
Files (x86)\Windows Kits\NETFXSDK\4.8\include\um" "-IC:\Program Files (x86)\Windows
Kits\10\include\10.0.18362.0\ucrt" "-IC:\Program Files (x86)\Windows
Kits\10\include\10.0.18362.0\shared" "-IC:\Program Files (x86)\Windows Kits\10\include\10.0.18362.0\um"
"-IC:\Program Files (x86)\Windows Kits\10\include\10.0.18362.0\winrt" "-IC:\Program Files (x86)\Windows
Kits\10\include\10.0.18362.0\cppwinrt" /EHsc /TpbtrievePython_wrap.cxx /Fobuild\temp.win-amd64-cpython-
310\Release\btrievePython_wrap.obj
btrievePython_wrap.cxx
creating C:\Develop\Python310\MyPrograms\build\lib.win-amd64-cpython-310
"C:\Program Files (x86)\Microsoft Visual
Studio\2019\Professional\VC\Tools\MSVC\14.29.30133\bin\HostX86\x64\link.exe" /nologo /INCREMENTAL:NO
/LTCG /DLL /MANIFEST:EMBED,ID=2 /MANIFESTUAC:NO "/LIBPATH:c:\Develop\Python310\MyPrograms\ "
/LIBPATH:C:\Develop\Python310\libs /LIBPATH:C:\Develop\Python310
/LIBPATH:C:\Develop\Python310\PCbuild\amd64 "/LIBPATH:C:\Program Files (x86)\Microsoft Visual
Studio\2019\Professional\VC\Tools\MSVC\14.29.30133\ATLMFC\lib\x64" "/LIBPATH:C:\Program Files
(x86)\Microsoft Visual Studio\2019\Professional\VC\Tools\MSVC\14.29.30133\lib\x64" "/LIBPATH:C:\Program
Files (x86)\Windows Kits\NETFXSDK\4.8\lib\um\x64" "/LIBPATH:C:\Program Files (x86)\Windows
Kits\10\lib\10.0.18362.0\ucrt\x64" "/LIBPATH:C:\Program Files (x86)\Windows
Kits\10\lib\10.0.18362.0\um\x64" btrieveCpp.lib /EXPORT:PyInit__btrievePython build\temp.win-amd64-
cpython-310\Release\btrievePython_wrap.obj /OUT:build\lib.win-amd64-cpython-310\_btrievePython.cp310-
win_amd64.pyd /IMPLIB:build\temp.win-amd64-cpython-310\Release\_btrievePython.cp310-win_amd64.lib
   Creating library build\temp.win-amd64-cpython-310\Release\_btrievePython.cp310-win_amd64.lib and
object build\temp.win-amd64-cpython-310\Release\_btrievePython.cp310-win_amd64.exp
Generating code
Finished generating code
```

If you are still having problems, then you might need to further review text inside of the scripts, specifically these files:

```
C:\Develop\Python310\Lib\distutils\ccompiler.py
C:\Develop\Python310\Lib\distutils\_msvccompiler.py
C:\Develop\Python310\Lib\distutils\msvc9compiler.py
```

If you do find yourself going down this road, you can try adding the statement "`log.set_verbosity(2)`" inside `find_vcvarsall()` (or other functions that are giving you fits), and the system will spit out all of the DEBUG messages as its runs.

Creating your Python Application

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

```
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, 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.

Special Note about Python 3.8: In v3.8, DLLs are loaded in a more restrictive manner – the operating system PATH is **not** used by default. The workaround is to have the following lines **before** the "import btrievePython" line:

```
import os
os.add_dll_directory("C:/Program Files/Actian/Zen/Bin")
```

Understanding Record Formats

One obvious question that arises when you look at this code is about the record formats. You can find the complete definition in the struct.pack and struct.unpack function documentation, but here are a few codes to get you started:

- b: 8-bit signed integer (byte)
- B: 8-bit unsigned integer (Byte)
- h: 16-bit signed integer (half-word)
- H: 16-bit unsigned integer (Half-word)
- i: 32-bit signed integer (integer)
- I: 32-bit unsigned integer (Integer)
- q: 64-bit signed integer (quad-word)
- Q: 64-bit unsigned integer (Quad-word)
- f: 32-bit floating point value (float)
- d: 64-bit floating point value (double)
- ?: 1-bit boolean
- s: character or byte array (string)

Further, there are indicators for defining the storage format of integers, also known as "endianness":

- <: Little-endian storage format
- >: Big-endian storage format

Finally, for multiple occurrences of a given field, simply include the repeat value as one or more digits before it – so "32s" indicates a 32-byte string.

Running Your Python Application

Running the application is even easier. Simply launch Python and pass in the script name!

```
python 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
os.add_dll_directory("C:/Program Files/Actian/Zen/Bin")
import sys
import struct
import btrievePython as btrv

btrieveFileName = "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))
```