Accessing Zen v13 from C++ on Windows Using the Btrieve 2 Interface

A White Paper From

GOLDSTAR SOFTWARE

www.GoldstarSoftware.com

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

Accessing Zen v13 from C++ on Windows Using the Btrieve 2 Interface

Last Updated: January 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.

While the older Btrieve API relied on only a single function call with 7 parameters, this simplicity put the onus on the developer to ensure that all of the data was correct, properly handling data buffers, key buffers, position blocks, key numbers, and more. Even worse, some of the functions, such as the GetNextExtended and Create operations, required esoteric buffers with perfect byte alignment, often causing developers great consternation and hair-pulling.

With the release of Zen v13, the new Btrieve 2 API is now available, and we fully expect it to become a favorite for developers. While the new API can be called from a myriad of different environments, such as Python, Perl, C#, and more, you can also use lower-level languages like C++ to gain the ultimate in flexibility and performance, with very little overhead and (more importantly) learning curve.

This paper is broken up into each of the steps you need to follow in order to build a simple C++ application on Windows. The example is a bit contrived – we open up the Person database file, read records one by one, and then count up the number of records where the birth date field is in March. However, this is only done to keep the sample code easy to understand. Because the Btrieve 2 API offers access to every Btrieve operation and the entire database, you can build extremely powerful applications and tools using these simple building blocks!

Note that while the code in Appendix A contains examples of a few 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 http://docs.actian.com/psql/psqlv13/btrieve2api/wwhelp/wwhimpl/common/html/wwhelp.htm#href=btrieve2api.htm&single=true.

Important Pre-requisites

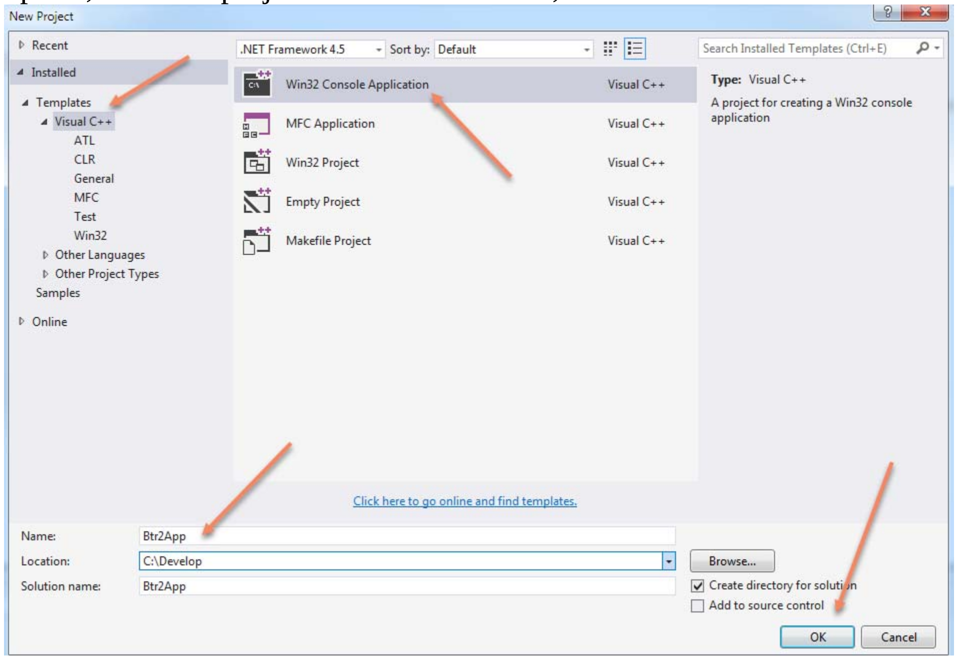
The following pre-requisites should be in place before you start.

1) You should have a Windows operating system properly installed.
2) You should have Microsoft Visual Studio properly installed. Screenshots within this paper are from VS2013, but the requirements are similar for both newer and older versions, so it should work with whichever developer components you have available, and extending this to other compilers should also be fairly straightforward.
3) You should have Zen (PSQL) v13 installed and running. If you are working on a stand-alone development machine, you can install either the Workgroup Engine or the Server Engine. (Both will work just fine with a 30-day trial.) If you are working within a shared environment, you should have a Zen v13 Workgroup Engine or Server Engine installed where the database files are located, and the Zen v13 Client installed on your development computer.

Creating a New Visual Studio Project

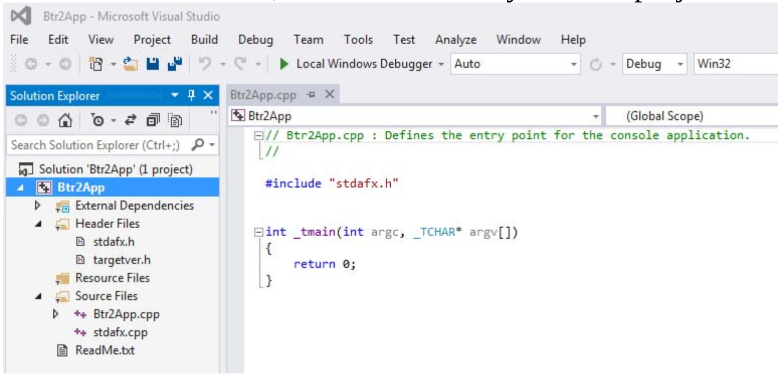
The first step is to create a new project within Visual Studio with basic starter code, and then we will configure it for a Btrieve 2 API application. In this example, we will build a simple command-line tool, so let's use the Wizard to build our project files:

1) Start Visual Studio.
2) Select **File/New/Project**….
3) Specify the new project details. We're building a **Win32 Console Application**, so select that option, enter the project name and location, and click OK.



Name: Btr2App
Location: C:\Develop
Solution name: Btr2App
☑ Create directory for solution
☐ Add to source control

4) From the next screen, click **Finish** and your new project will be opened with some starter code:



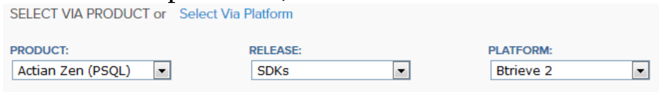
```
// Btr2App.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"


int _tmain(int argc, _TCHAR* argv[])
{
	return 0;
}
```

5) If you wish, you can build this program and it will run (though it won't do anything yet).

Downloading and Installing the Btrieve 2 SDK

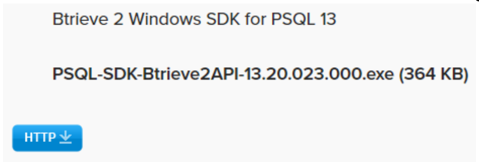
The next piece you need is the SDK download for Btrieve 2:

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



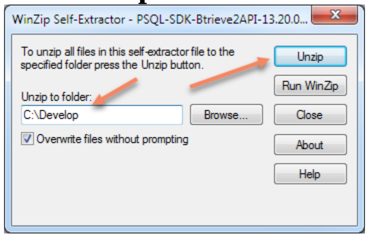
PRODUCT:	RELEASE:	PLATFORM:
Actian Zen (PSQL)	SDKs	Btrieve 2

3) Scroll down and open up the link for **Btrieve 2**, then click on the **HTTP** button to download the Btrieve 2 Windows SDK for PSQL 13. (Note that the version may change in the future.)


Btrieve 2 Windows SDK for PSQL 13

PSQL-SDK-Btrieve2API-13.20.023.000.exe (364 KB)

4) After it downloads, double-click the EXE file to launch the installer.
5) Change the **Unzip to Folder** to your project base folder (**C:\Develop\** in my example**)** and click **Unzip**:


WinZip Self-Extractor - PSQL-SDK-Btrieve2API-13.20.0...

To unzip all files in this self-extractor file to the specified folder press the Unzip button.

Unzip to folder: C:\Develop

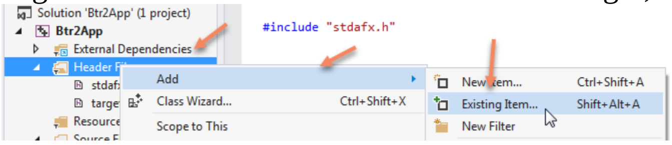
☑ Overwrite files without prompting

6) You will now have the Btrieve 2 API components in a folder on your workstation. Although the actual folder name can change as the versions are updated, it is currently called `c:\Develop\PSQL-SDK-Btrieve2API-13.20.023.000`
7) Copy the **btrieveCpp.h** and **btrieveC.h** file from the "include" folder to your project folder where the source code is located. (In my example, this is C:\Develop\Btr2App\Btr2App.)
8) Copy the **btrieveCpp.lib** files from the "win32\x86" folder to your project folder also.

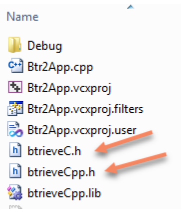
Configuring your Visual Studio Project

Now that the components are available, we need to add the header file and library to our project:

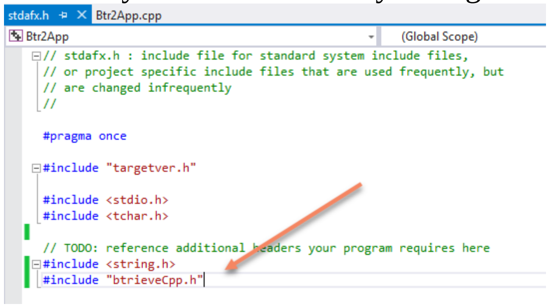
1) Right-click on the **Header Files** in the left margin, select **Add**, then **Existing Item**.


Solution 'Btr2App' (1 project)
Btr2App
External Dependencies
Header Files
stdafx
targe
Resource
Source F
#include "stdafx.h"
Add
Class Wizard... Ctrl+Shift+X
Scope to This
New Item... Ctrl+Shift+A
Existing Item... Shift+Alt+A
New Filter

2) Select **btrieveC.h** and **btrieveCpp.h** and click **Add**:

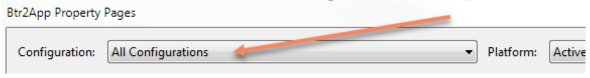


3) Open the stdafx.h file and add the include line for the **"btrieveCpp.h"** file. This is a good time to add any other header files you might need, such as **<string.h>** or **<malloc.h>**.



```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#pragma once

#include "targetver.h"

#include <stdio.h>
#include <tchar.h>

// TODO: reference additional headers your program requires here
#include <string.h>
#include "btrieveCpp.h"
```

4) Open the project configuration by selecting **Project** from the menu, then **Properties**.
5) Make sure that the **Configuration** option is set to **All Configurations**:

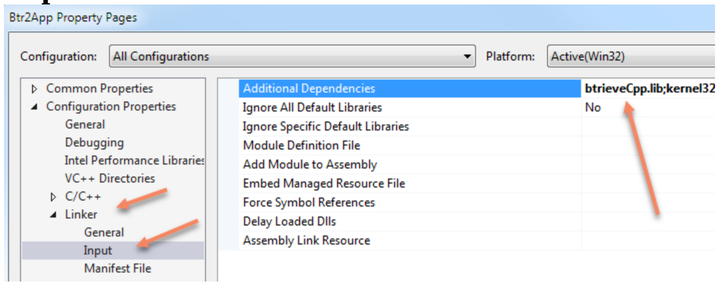


Btr2App Property Pages

Configuration: All Configurations

Platform: Active

6) Open up the **Linker** options and select **Input**, then add "**btrieveCpp.lib;**" to the **Additional Dependencies** line and click OK**:**



Btr2App Property Pages

Configuration: All Configurations

Platform: Active(Win32)

Additional Dependencies	btrieveCpp.lib;kernel32
Ignore All Default Libraries	No
Ignore Specific Default Libraries	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded Dlls	
Assembly Link Resource	

Creating your C++ Application

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

- Opening a File
- Reading the First Record
- Reading the Next Record
- Analyzing Data Inside Each Record
- Closing the File
- Resetting the Database Connection

We're going to build the sample application first, so copy the text from Appendix A and paste it into the file Btr2App.cpp. (You can replace the entire contents of the existing file.)

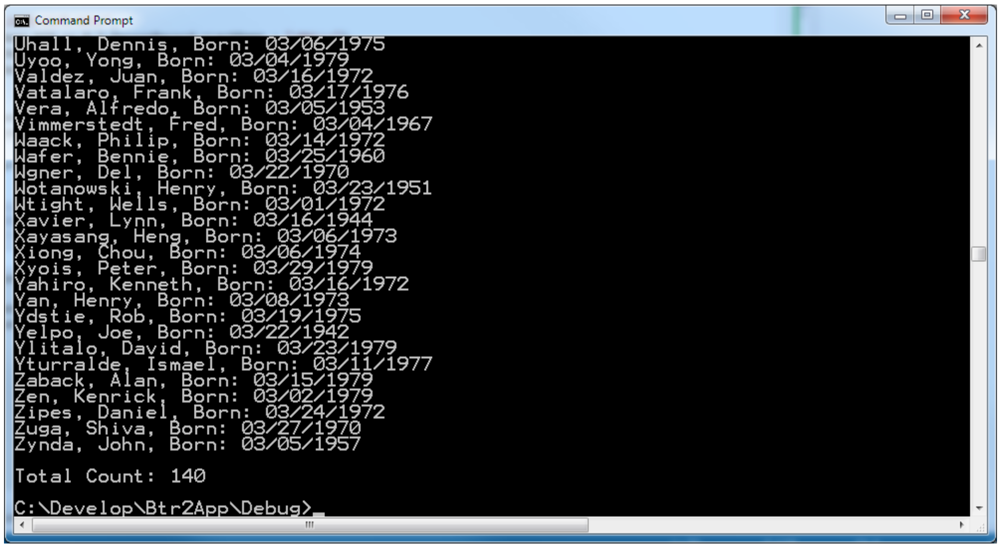
Before we compile and run the code, let's review a few important elements in this code:

- Lines 5 and 25 are needed because most Btrieve data structures are byte-aligned, and the compiler attempts to enable word alignment by default. You could either use the **#pragma pack** operations to do this, or you can change the **Struct Member Alignment** setting in the **C Code Generation** settings of your project. If you forget this important detail, then your fields may not align correctly with the data structure, and the buffer sizes will differ as well!
- Line 13 starts the definition of the *Person* Record. Since we are not accessing every field, I have only defined the fields that we are going to access. Your own code would, of course, define EVERY field in the data structure.
- Lines 16, 18, and 21 contain Null Indicator Bytes (NIBs). The following fields support true nulls, which PSQL implements with a NIB. The data value is ONLY valid when the NIB is equal to 0, otherwise the value is NULL. Many older applications do not use NIBs.
- Lines 41-45 open the target database file and quit if an error is returned.
- Lines 47-52 read the first record of the table on Key 1, which is the *LastName*/*FirstName* field combination. Strangely, Actian defines the constant INDEX_1 as 0, INDEX_2 as 1, and so on. This seems VERY silly to me, and it is unclear why they chose this nomenclature. Note that the call to *RecordRetrieveFirst* actually returns the length of the record read, not a status code. Because of this, we check for a -1 (indicating that some error occurred), then get and display the status code before returning.

- Line 62 verifies that the month is March (3). Note that we ALSO check the NIB, just in case the *BirthDate* field is actually NULL.
- Lines 64-68 display the person's name and birthdate and tally a counter. We have to watch out for NULL values, but the in-line conditional handles this quite succinctly.
- Lines 70-71 read the next record from the database file. Like *RecordRetrieveFirst* above, *RecordRetrieveNext* returns the record length. In this case, we don't care about the record length, but we DO want the status code so that we know when we can terminate the loop, so we get the status code every time. Any non-zero status code will terminate the loop. A thorough developer would check for Status 9 (EndOfFile) and display a message if some other error was returned, but this is an exercise left for the reader.
- Line 77 closes the database file that we opened.
- Line 79 resets the database connection, releasing all locks, file handles, and other resources. You should **always** do this before your application terminates.

Running Your C++ Application

Build the application using the **Build** button or menu in Visual Studio. To test it, open a **Command Prompt** window, go to the folder where your new EXE file resides, and run it. You should get a result similar to that shown here:



```
Uhall, Dennis, Born: 03/06/1975
Uyoo, Yong, Born: 03/04/1979
Valdez, Juan, Born: 03/16/1972
Vatalaro, Frank, Born: 03/17/1976
Vera, Alfredo, Born: 03/05/1953
Vimmerstedt, Fred, Born: 03/04/1967
Waack, Philip, Born: 03/14/1972
Wafer, Bennie, Born: 03/25/1960
Wgner, Del, Born: 03/22/1970
Wotanowski, Henry, Born: 03/23/1951
Wtight, Wells, Born: 03/01/1972
Xavier, Lynn, Born: 03/16/1944
Xayasang, Heng, Born: 03/06/1973
Xiong, Chou, Born: 03/06/1974
Xyois, Peter, Born: 03/29/1979
Yahiro, Kenneth, Born: 03/16/1972
Yan, Henry, Born: 03/08/1973
Ydstie, Rob, Born: 03/19/1975
Yelpo, Joe, Born: 03/22/1942
Ylitalo, David, Born: 03/23/1979
Yturralde, Ismael, Born: 03/11/1977
Zaback, Alan, Born: 03/15/1979
Zen, Kenrick, Born: 03/02/1979
Zipes, Daniel, Born: 03/24/1972
Zuga, Shiva, Born: 03/27/1970
Zynda, John, Born: 03/05/1957

Total Count: 140

C:\Develop\Btr2App\Debug>_
```

To verify that it is working correctly, use the PCC to edit some of the data records in the Demodata.Person table (remember that only March birthdates will be displayed) and re-run your application. You should see the data set changing accordingly. You have now successfully written your own Btrieve 2 application!

Expanding Your Knowledge

Want to know if you understand what is going on? Try these modifications:

1) Change the program to read the data by Key 2 instead of Key 1. Which line do you need to change? Can you figure out in what order the records are being displayed now?
2) Change the selection criteria to pickup only those records where the *Perm_State* field is "TX". You will need to expand the data structure to add the *Perm_Street*, *Perm_City*, and *Perm_State* fields. You can check the Person table definition in the PCC for field lengths, or you can trust

me that they are VARCHAR fields of 30, 30, and 2 bytes, respectively. (Remember, though, that a VARCHAR field is a null-terminated string, so be sure to account for the null at the end!)

3) Update the printf in Line 64 to also display the *Perm_State* and *Perm_City* fields. Did the data display properly? If you are seeing fields getting cut off, check your alignment, and verify that you remembered to add the NIB fields in front of each of the new fields you created in #2!
4) If you did all of the above code changes, you may notice that the application is reading all 1500 records from the *Person* table to display a much smaller set of records. However, since the data is sorted by the *State/City* key, why bother reading the records at the beginning of the database that will never match the State restriction? Change your application to use the *RecordRetrieve* operation to implement a *GetGreaterOrEqual* call instead. You'll need to specify a properly-defined key buffer (with NIBs) and seed your data with the *Perm_State* field of TX.
5) If you were paying attention, you'll see that the program now much faster, because it is doing much less work. However, as written, the code still runs until ends of file, meaning that records from Utah, Wyoming, etc. are also being read, even though they could never match the restriction. As another optimization, change your code to terminate the loop once the last TX record has been read.
6) **For expert developers only:** After the *RecordRetrieve* operation, try calling the *BulkRetrieveNext* operation instead, which will allow you to read the entire TX data set in just two function calls! Add a filter to limit the result set, and add fields to pull only the fields you want as part of your report. This combination will leverage the true power of client/server computing by moving the workload to the server and reducing the number of round trips from your application to the engine. The performance gains, especially on a slow network, might just surprise you!

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

Finding More Help

If you have other problems getting this to work, you can 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 through our Developer Jump Start Program. 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.

```
// Btr2App.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"

#pragma pack(push, 1)                                                      // Set up Byte alignment for these structures
typedef struct BDateRec                                                    // This structure allows us to parse the date easier.
{
          unsigned char Day;
          unsigned char Month;
          unsigned short Year;
} BDATEREC;

typedef struct PersonRecord                                                // This definition was created by examining the definition for the Person table.
{                                                                          // We are reading a limited subset of fields, so we can get cheat a bit here
          long long           Id;                                          // and define ONLY the fields we care about.
          unsigned char       FirstName_NIB;                               // In most cases, you would define the entire structure, of course.
          char                FirstName[16];
          unsigned char       LastName_NIB;
          char                LastName[26];
          char                filler[211];
          unsigned char       BirthDate_NIB;
          BDATEREC            BirthDate;
          // There is mmore, but we don't need anything after this field....
} PERSONREC;
#pragma pack(pop)                                                          // Back to normal alignment again

int _tmain(int argc, _TCHAR* argv[])
{
          char fileName[255];
          BtrieveClient btrieveClient(0x4232, 0);
          Btrieve::StatusCode status = Btrieve::STATUS_CODE_UNKNOWN;
          BtrieveFile btrieveFile;
          unsigned char RecordData[BTRIEVE_MAXIMUM_RECORD_LENGTH];
          PERSONREC *PersonRecord;
          int BornInMarch;

          // The PERSON.MKD file is part of the sample data installed by default with the PSQL engine.
          strcpy_s(fileName, 255, "C:\\ProgramData\\Actian\\PSQL\\Demodata\\Person.mkd");

          // Open the database file
          if ((status = btrieveClient.FileOpen(&btrieveFile, fileName, NULL, Btrieve::OPEN_MODE_NORMAL)) != Btrieve::STATUS_CODE_NO_ERROR)
          {
                    printf("Error opening file %s: Status %d:%s.\n", fileName, status, Btrieve::StatusCodeToString(status));
                    return(-1);
          }

          if (btrieveFile.RecordRetrieveFirst(Btrieve::INDEX_2, (char *)RecordData, BTRIEVE_MAXIMUM_RECORD_LENGTH) == -1)
          {
                    status = btrieveFile.GetLastStatusCode();
                    printf("Error reading the first record, Status %d:%s.\n", status, Btrieve::StatusCodeToString(status));
                    return(-1);
          }
          else
          {
                    // Let's initialize some variables that we'll be using in the loop
                    BornInMarch = 0;
                    PersonRecord = (PERSONREC *)RecordData;                                  // We do this just to make accessing the data easier
                    status = Btrieve::STATUS_CODE_NO_ERROR;                                  // Initialize the status code for my loop
                    while (status == Btrieve::STATUS_CODE_NO_ERROR)
                    {
                              // If the Person was born in March, display their name and birth date and add one to the counter.
                              if (PersonRecord->BirthDate_NIB == 0 && PersonRecord->BirthDate.Month == 3)
                              {
                                        printf("%s, %s, Born: %.2d/%.2d/%.4d\n",
                                                  (PersonRecord->LastName_NIB == 0 ? PersonRecord->LastName : "<NULL>"),
                                                  (PersonRecord->FirstName_NIB == 0 ? PersonRecord->FirstName : "<NULL>"),
                                                  PersonRecord->BirthDate.Month, PersonRecord->BirthDate.Day, PersonRecord->BirthDate.Year);
                                        BornInMarch++;
                              }
                              btrieveFile.RecordRetrieveNext((char *)RecordData, BTRIEVE_MAXIMUM_RECORD_LENGTH);
                              status = btrieveFile.GetLastStatusCode();
                    }
                    printf("\nTotal Count: %d\n", BornInMarch);
          }

          // Close the database file. I'm skipping the error handler, because we're quitting anyway.
          btrieveClient.FileClose(&btrieveFile);
          // Shut down the database connection, because mom taught me to clean up after myself.
          btrieveClient.Reset();
          return(0);
}
```