Getting Started with the Btrieve API within Zen v14 From Visual Studio C

A White Paper From

GOLDSTAR SOFTWARE

www.GoldstarSoftware.com

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

Getting Started with the Btrieve API within Zen v14 from Visual Studio C

Last Updated: 01/26/2022

The hardest part about learning a new development environment, whether it be a new language, GUI, API, database, or just about anything else, is locating the "right" starting point to get your project kicked off. This is quite like the "blank page" syndrome some writers feel when starting a new project.

In most books about a new language, this is addressed by the obligatory "Hello World" application, which serves as an introduction to the environment as a whole, including the proper syntax to write, compile, and execute your applications.

In that same vein, this simple step-by-step guide was created to show you each piece of the environment needed to start a new Visual Studio C project accessing the Btrieve API. One of the major issues with such a document is that the user interface changes over time, making the guide useless to a true beginner. To that end, please be aware that some UI elements here may not match exactly what you see, but you should be able to track down the right spot with a bit of hunting.

Prerequisites

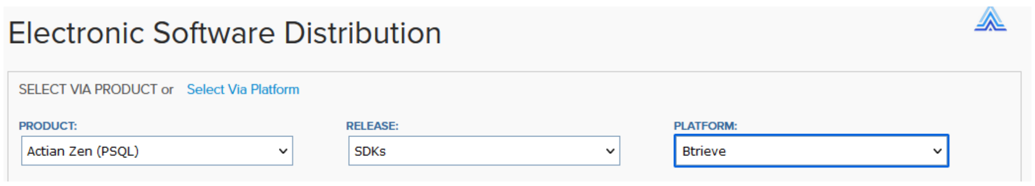
This document assumes that you already have the development environment and database engine installed and functioning. Specifically, this guide contains UI images from Visual Studio 2019 and Actian Zen v14, but whatever version you have should at least be similar enough to follow along. If you need help installing these two components, then refer to the documentation for each component as needed.

Downloading and Installing the Btrieve SDK

In addition to the development environment, you will need the Btrieve SDK download from Actian. As of this writing, you can find this file at:

https://esd.actian.com/product/Zen_PSQL

Select the **SDK** release and **Btrieve** platform as shown here:



Electronic Software Distribution

SELECT VIA PRODUCT or Select Via Platform

PRODUCT: Actian Zen (PSQL)

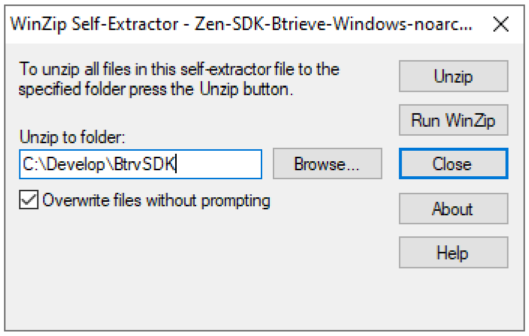
RELEASE: SDKs

PLATFORM: Btrieve

Then from the list of downloads available, download the Btrieve component, which typically starts with the text "Zen-SDK-Btrieve-Windows". As of this writing, the file is fully named:

Zen-SDK-Btrieve-Windows-noarch-14.20.012.000.exe

When you run this file, you will get an Extraction dialog box that looks like this:



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

Unzip to folder: C:\Develop\BtrvSDK

☑ Overwrite files without prompting

Provide a proper path for the files to be extracted and remember where you put them! Some people like to explode these files into each project to keep everything complete. However, upgrading a large number of projects to new SDK components gets complicated, so I prefer to keep everything in one centralized folder for inclusion into each project from there.

This will create two folders, DOC and INTF, in the target folder. The DOC folder contains documentation on the API calls in PDF format. The INTF folder, on the other hand, contains the real "meat" of the interface components, including subfolders for C, CBuilder, Delphi, Pascal, and VB.Net2010. There is one other critical folder called IMPLIB that contains the import libraries, which we will use when linking.

For this project, we are going to work on a C application, so let's drill a bit further down into the C folder and see what is in each file, in alphabetical order:

- **BLOBHDR.H**: This file contains the definitions of constants and data structures needed for Chunk operations. Chunk operations are a bit complicated, and will not be covered here, but it is far easier to use these definitions than to attempt to write your own.
- **BTISTRUCT.H**: This file contains other structure definitions and constants for the CREATE and STATEXTENDED operations, among others.
- **BTITYPES.H**: This file contains platform-independent type definitions for the various Btrieve data types, which is important as some compilers use different definitions for the same types (e.g. "long long" = "LARGE_INTEGER" = "signed __int64"). This header file makes it possible to simply use BTI_LONGLONG or BTI_INT64 in your own code, and let the translation occur in the header.
- **BTRAPI.C**: This C source code module actually provides the definition for the BTRV and BTRVID functions, which handle some of the extra load required to call BTRCALL and BTRCALLID by simply wrapping these operations to make them easier to use.
- **BTRAPI.H**: This header file provides the definitions for the functions in BTRAPI.C and should be included in your own code.
- **BTRCONST.H**: Another important header file you should include directly into your application, this one defines many of the commonly-used constants, such as the operation codes, file open modes, status (return) codes, key type numbers, and

more. While you certainly do not need to leverage these constants, doing so can make your code much more readable.

- **BTRSAMP.C**: This is Actian's equivalent of the Hello World application, this sample code shows you how to do some of the basic operations, such as creating a Client ID, making a Version call, opening a file, reading a record, creating a new file based on the old one, reading records with a GetNextExtended call, and inserting records into the newly-created file.
- **BTRVEXID.H**: This header file is new for PSQL v13 and newer, and provides the interface definitions for the BTRVEX and BTRVEXID functions. These functions are available ONLY in v13 and newer, but are required if you intend to access any database files (in v13 format) that contain more than 4 billion records or are larger than 256GB in size.
- **LOGINAPI.C**: Another sample file, this one demonstrates how to use the Btrieve Login API function.
- **MAKEFILE**: This makefile is provided for older command-line compilers, so you might never need this.

The last file you will need for your application is located in the IMPLIB folder. The file W3BTRV7.LIB will contain the library definitions from the core interface DLL (W3BTRV7.DLL) which is linked into your application to provide the BTRCALL functions used to access the Btrieve API from 32-bit applications. If you are writing a 64-bit application, then you will instead use the W64BTRV.LIB and DLL.

AN IMPORTANT NOTE ABOUT PACKING

If you look at the header file BTISTRUCT.H, you will see that it repeatedly wraps each structure definition inside a **#pragma pack(1)** statement. This is done because most Btrieve structures are packed with single-byte alignment, whereas the default compiler option may be to pack on a 4-byte boundary. This can be better explained with the following structure definition:

```
typedef struct
{
   BTI_UINT32 Value1;
   BTI_BYTE Value2;
   BTI_UINT32 Value3;
} MY_DATA_STRUCTURE;
```

With byte alignment (i.e. pack(1)), this data structure will be 9 bytes in length. However, with the default 32-bit alignment (i.e. pack(4)), the compiler will insert 3 bytes of filler to align the Value3 variable on a longword boundary, and this structure will actually be 12 bytes long. This is done for performance reasons at the CPU level.

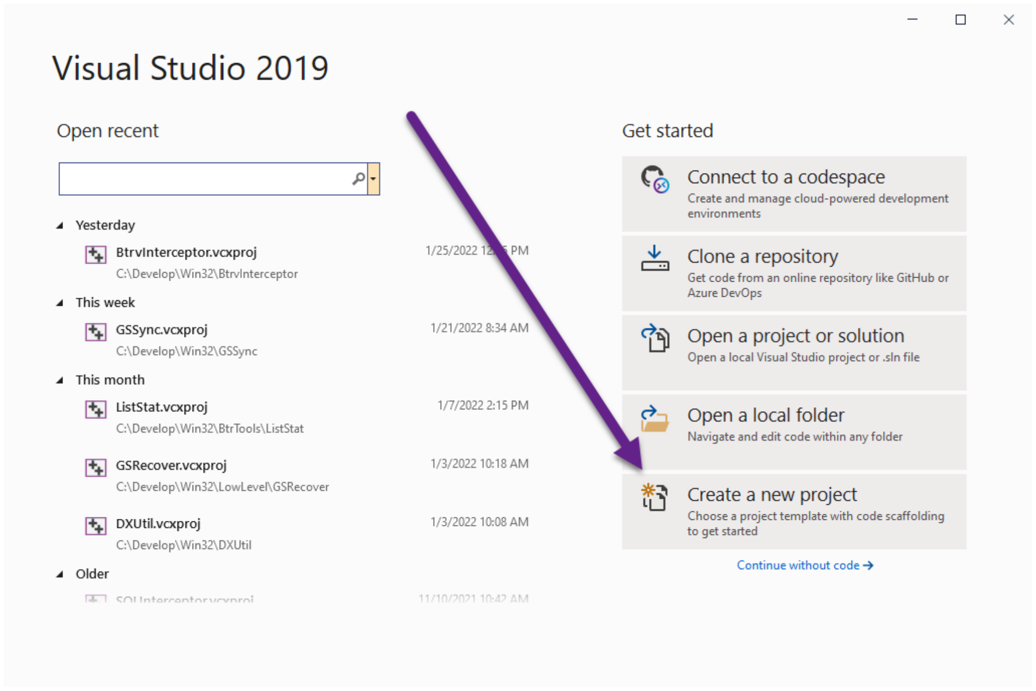
In itself, alignment is immaterial, but if you are trying to match an existing definition, you may find that the data is shifted when you read or write it, and this can cause much wailing and gnashing of teeth as nothing lines up correctly.

In short, be sure to force all of your own data structure definitions to pack(1) as well!

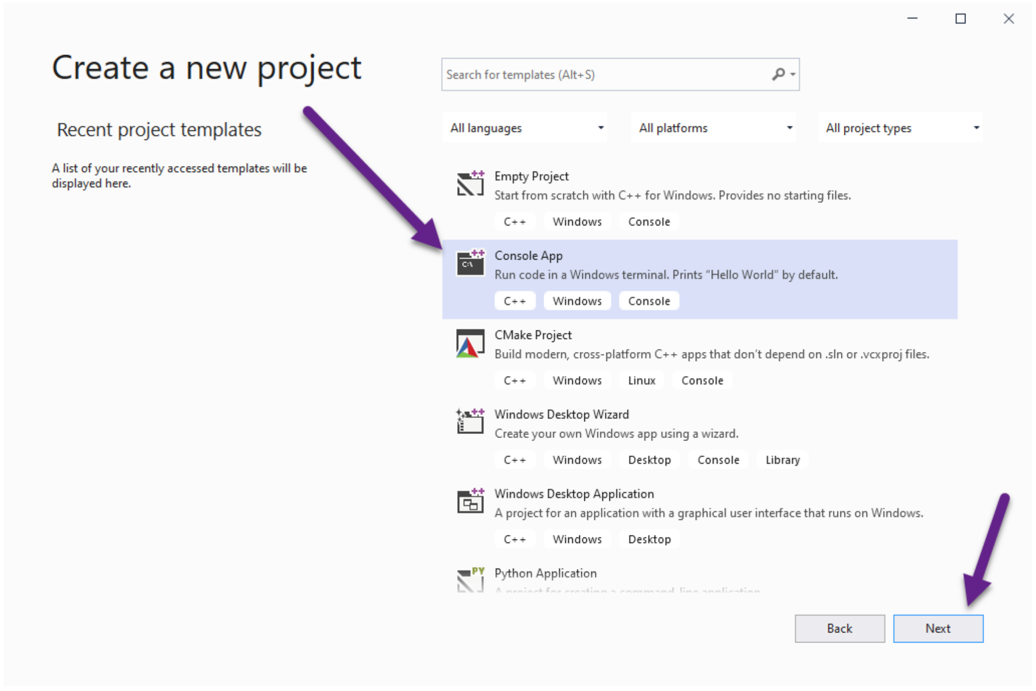
Now that we've got THAT out of the way, let's dig right in!

Creating a New Visual Studio Project

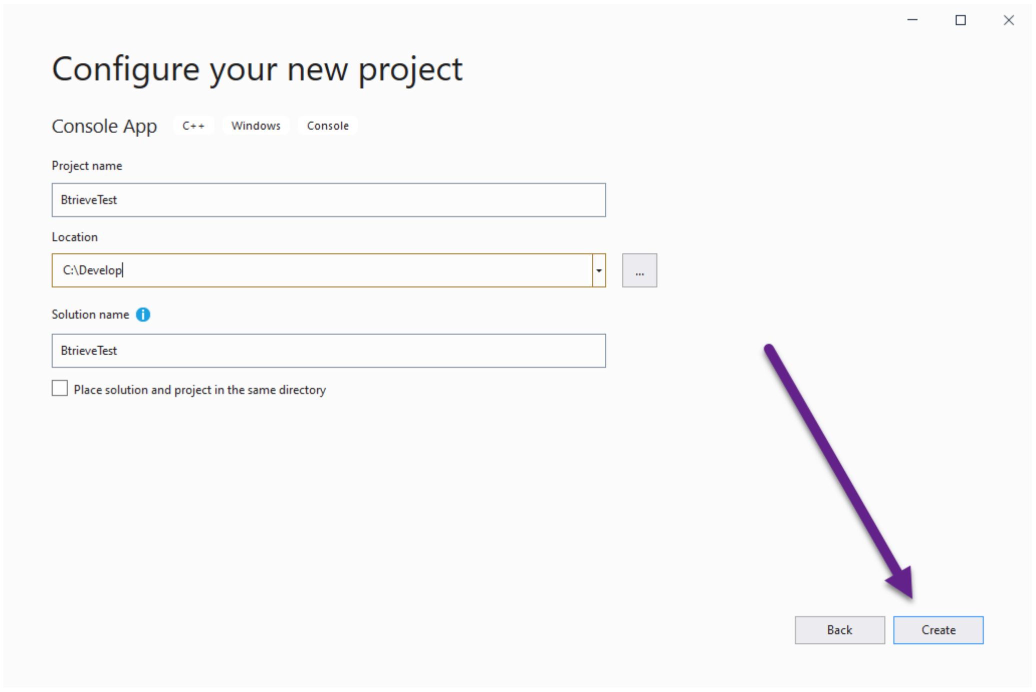
The first step to building our new application is to create a new project in Visual Studio.


Visual Studio 2019
Open recent
Yesterday
BtrvInterceptor.vcxproj
C:\Develop\Win32\BtrvInterceptor
1/25/2022 12:45 PM
This week
GSSync.vcxproj
C:\Develop\Win32\GSSync
1/21/2022 8:34 AM
This month
ListStat.vcxproj
C:\Develop\Win32\BtrTools\ListStat
1/7/2022 2:15 PM
GSRecover.vcxproj
C:\Develop\Win32\LowLevel\GSRecover
1/3/2022 10:18 AM
DXUtil.vcxproj
C:\Develop\Win32\DXUtil
1/3/2022 10:08 AM
Older
Get started
Connect to a codespace
Create and manage cloud-powered development environments
Clone a repository
Get code from an online repository like GitHub or Azure DevOps
Open a project or solution
Open a local Visual Studio project or .sln file
Open a local folder
Navigate and edit code within any folder
Create a new project
Choose a project template with code scaffolding to get started
Continue without code →

For this simple project, we are going to create a console application that be run directly from a command line, so we'll select this option next:


Create a new project
Recent project templates
A list of your recently accessed templates will be displayed here.
Search for templates (Alt+S)
All languages
All platforms
All project types
Empty Project
Start from scratch with C++ for Windows. Provides no starting files.
C++ Windows Console
Console App
Run code in a Windows terminal. Prints "Hello World" by default.
C++ Windows Console
CMake Project
Build modern, cross-platform C++ apps that don't depend on .sln or .vcxproj files.
C++ Windows Linux Console
Windows Desktop Wizard
Create your own Windows app using a wizard.
C++ Windows Desktop Console Library
Windows Desktop Application
A project for an application with a graphical user interface that runs on Windows.
C++ Windows Desktop
Python Application
Back
Next

We then name the project and create the project folder, then click Create:



Configure your new project

Console App C++ Windows Console

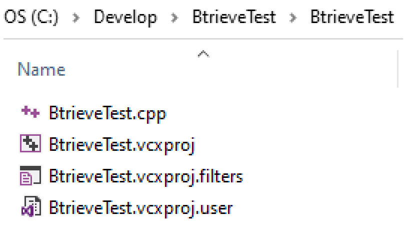
Project name: BtrieveTest

Location: C:\Develop

Solution name: BtrieveTest

☐ Place solution and project in the same directory

This will now create the project and a smattering of starting files, as shown here:

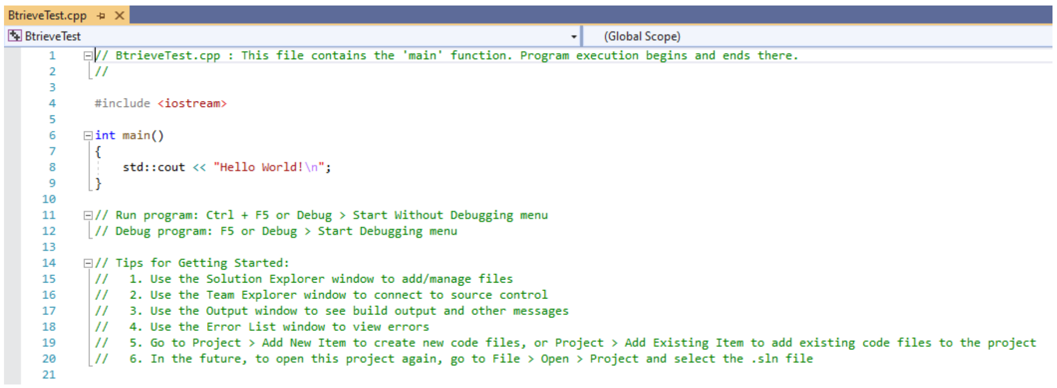


OS (C:) › Develop › BtrieveTest › BtrieveTest

Name

BtrieveTest.cpp
BtrieveTest.vcxproj
BtrieveTest.vcxproj.filters
BtrieveTest.vcxproj.user

The source code file, BtrieveTest.CPP, contains starter “hello world” code for you already:

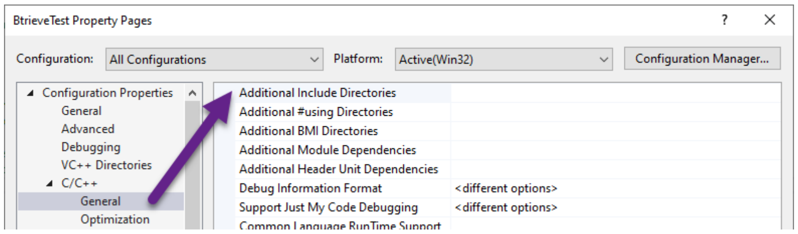


```cpp
// BtrieveTest.cpp : This file contains the 'main' function. Program execution begins and ends there.
//

#include <iostream>

int main()
{
    std::cout << "Hello World!\n";
}

// Run program: Ctrl + F5 or Debug > Start Without Debugging menu
// Debug program: F5 or Debug > Start Debugging menu

// Tips for Getting Started:
//   1. Use the Solution Explorer window to add/manage files
//   2. Use the Team Explorer window to connect to source control
//   3. Use the Output window to see build output and other messages
//   4. Use the Error List window to view errors
//   5. Go to Project > Add New Item to create new code files, or Project > Add Existing Item to add existing code files to the project
//   6. In the future, to open this project again, go to File > Open > Project and select the .sln file

```

In fact, you can (and probably should) compile this code immediately and verify that it works as expected.

Import Needed Files from the Btrieve SDK

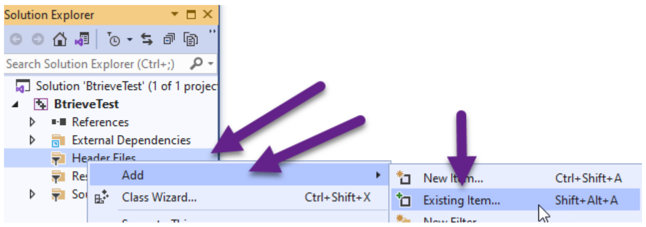
If you expect to be building many different projects, then you will be better off setting up your development environment with a single set of SDK files. This can be done by creating a folder at some higher level where the files can reside, or it can be done by adding the SDK folder to the the Additional Include Directories value in your project properties:


BtrieveTest Property Pages
Configuration: All Configurations
Platform: Active(Win32)
Configuration Manager...
Configuration Properties
General
Advanced
Debugging
VC++ Directories
C/C++
General
Optimization
Additional Include Directories
Additional #using Directories
Additional BMI Directories
Additional Module Dependencies
Additional Header Unit Dependencies
Debug Information Format <different options>
Support Just My Code Debugging <different options>
Common Language RunTime Support

Note that when configuring something like this, you should set the **Configuration** option to “All Configurations”. If you leave it set to Debug or Release, then you’ll quickly dfind that you have to make the change multiple times.

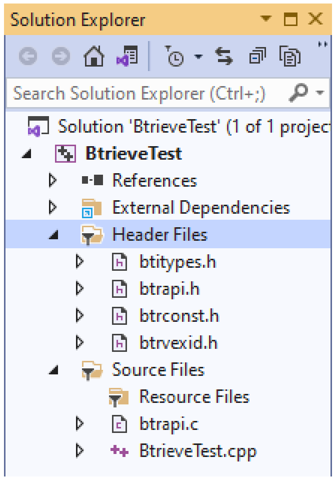
However, if you are building a single project, then it is simple enough to copy the needed files from the Btrieve SDK download into your project folder, and this is the solution we will use here.

1. First, copy the files you intend to use from the SDK folder into your project folder. In this case, we expect to use BTRCONST.H, BTITYPES.H, BTR, BTRVEXID.H and BTRAPI.H header files, as well as the BTRAPI.C source file.
2. Then, from the INTF folder, copy over the appropriate library file. IN this case, we are building a 32-bit application, so I’m bringing over W3BTRV7.DLL from the WIN32 folder.
3. In Visual Studio Solution Explorer, right-click the **Header Files** line and add each header file in turn:


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'BtrieveTest' (1 of 1 projec
BtrieveTest
References
External Dependencies
Header Files
Add
Class Wizard... Ctrl+Shift+X
New Item... Ctrl+Shift+A
Existing Item... Shift+Alt+A
New Filter

4. Do the same for the BTRAPI.C file in the **Source** folder.

You'll now have a solution that looks like this:

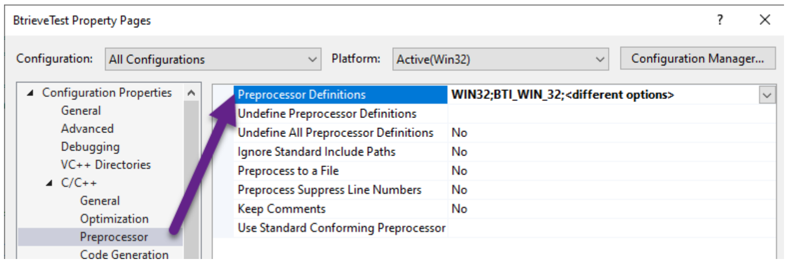


If you attempt to build the solution now, you will get an error about a missing directive. (Try it and see for yourself!) This brings us to the next step, configuring the project.

Configure the Project Compiler and Linker Options

With platform-independent header files, we have to tell the preprocessor which platform we are building for. While it may be tempting to simply add a **#define** right into the top of your source code, you should avoid doing this. At some point in the future, you may want to target a different environment (e.g. 64-bit Windows), and changing this in the compiler options is simply easier in the long run.

As we are building a 32-bit application, we need to define the compiler directive "`BTI_WIN_32`" so that the proper definitions are constructed at compile-time. To do this, open your project properties screen again, make sure that "All Configurations" is selected, and go to the C/C++ Preprocessor options, and add the definition to the top line:



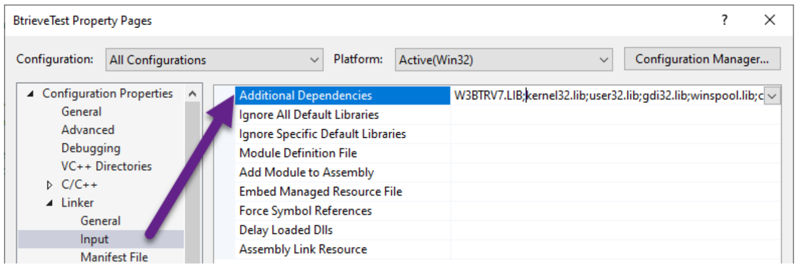
Preprocessor Definitions	WIN32;BTI_WIN_32;<different options>
Undefine Preprocessor Definitions	
Undefine All Preprocessor Definitions	No
Ignore Standard Include Paths	No
Preprocess to a File	No
Preprocess Suppress Line Numbers	No
Keep Comments	No
Use Standard Conforming Preprocessor	

Now, the code will compile successfully, but you will now get linker errors:

```
error LNK2019: unresolved external symbol _BTRCALL@28 referenced in function _BTRV@24
error LNK2019: unresolved external symbol _BTRCALLID@32 referenced in function _BTRVID@28
```

To fix this, we ALSO need to tell the compiler about the linker library.

1. Go back to the Project Properties page and verify that All Configurations is selected.
2. Open the Linker section and go to the Input page.
3. Add the LIB file to the Additional Dependencies screen.


BtrieveTest Property Pages

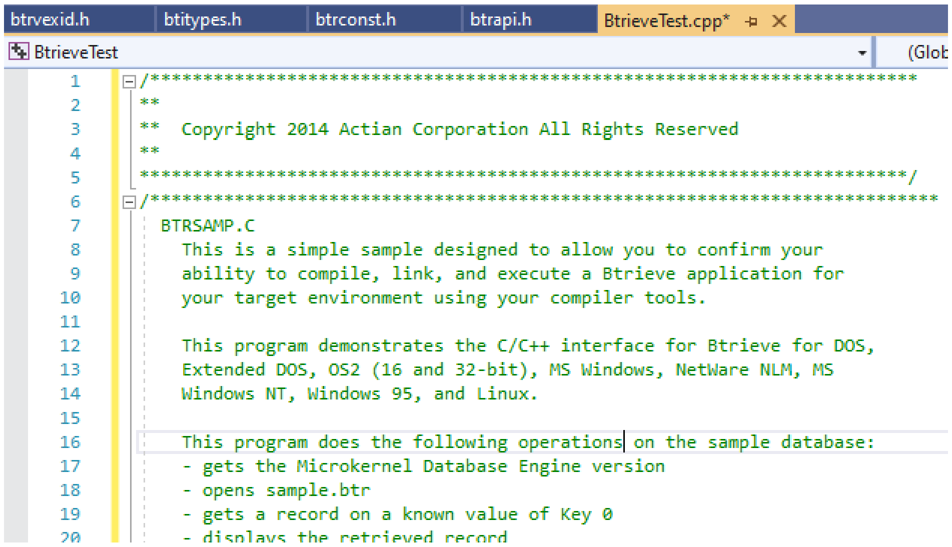
Configuration: All Configurations — Platform: Active(Win32) — Configuration Manager...

Additional Dependencies	W3BTRV7.LIB;kernel32.lib;user32.lib;gdi32.lib;winspool.lib;c
Ignore All Default Libraries	
Ignore Specific Default Libraries	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded Dlls	
Assembly Link Resource	

NOW, you can compile and run your program, and it will work! Of course, it only outputs a simple “Hello World” message and does nothing with the database, but that’s the majority of what is needed to get VS configured.

Compile the Sample Btrieve Code

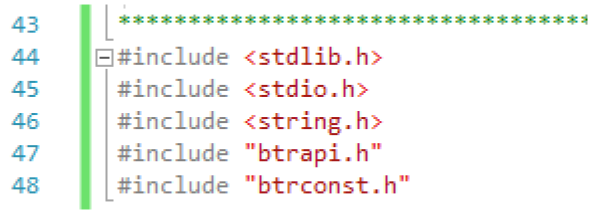
Once we have the environment ready, we can compile the sample code. IN this case, we are going to take a shortcut and simply coipy ALL of the text from the BTRSAMP.C file provided in the SDK into out BtrieveTest.CPP file in Visual Studio. This will now look like the following:



```
/*************************************************************************
**
**  Copyright 2014 Actian Corporation All Rights Reserved
**
*************************************************************************/
/*************************************************************************
  BTRSAMP.C
    This is a simple sample designed to allow you to confirm your
    ability to compile, link, and execute a Btrieve application for
    your target environment using your compiler tools.

    This program demonstrates the C/C++ interface for Btrieve for DOS,
    Extended DOS, OS2 (16 and 32-bit), MS Windows, NetWare NLM, MS
    Windows NT, Windows 95, and Linux.

    This program does the following operations on the sample database:
    - gets the Microkernel Database Engine version
    - opens sample.btr
    - gets a record on a known value of Key 0
    - displays the retrieved record
```

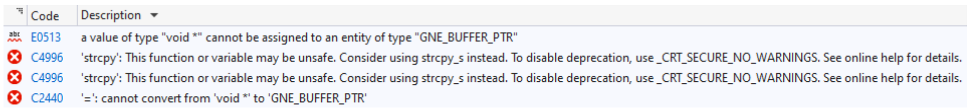
If you’re impatient, you might try to immediately compile the new code. If you took the extra time to add your header files into the Include path, then you may be delighted to find that it works. However, if you are following these instructions, you’ll get a boat-load of undefined identifier messages.

The cause of these errors is in the way the header files are included, with angle brackets, which assume they are in the standard include path. Change lines 47 and 48 to use quotes instead:



```
43   ***********************************
44   #include <stdlib.h>
45   #include <stdio.h>
46   #include <string.h>
47   #include "btrapi.h"
48   #include "btrconst.h"
```

Now, everything SHOULD work. However, when we compile and link on Visual Studio 2019, we still get 4 error messages:



Code	Description
E0513	a value of type "void *" cannot be assigned to an entity of type "GNE_BUFFER_PTR"
C4996	'strcpy': This function or variable may be unsafe. Consider using strcpy_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details.
C4996	'strcpy': This function or variable may be unsafe. Consider using strcpy_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details.
C2440	'=': cannot convert from 'void *' to 'GNE_BUFFER_PTR'

While this code worked originally when Actian built the sample code, Microsoft made some changes in more recent Visual Studio editions which broke a few things.

The first change makes it impossible to cast a VOID * to a pointer of another type. We fix this by adding a forced cast to the GNE_BUFFER_PTR in line 441:

```
gneBuffer = (GNE_BUFFER_PTR) malloc(sizeof(GNE_BUFFER));
```

This leaves us with the two "unsafe" messages, which stem from the old string handling library. Back in the "good old days", you simply called **strcpy()** to copy a string, and the string library copied bytes until the first NULL byte was seen. However, if you didn't allocate a large enough target buffer, this could overwrite memory or cause access violations. In an attempt to force developers to be more cognizant of memory errors (and make code more stable), Microsoft created new functions (like **strcpy_s()**) that ALSO require the target string length as a paramater. These functions validate the data as it is being copied to prevent a memory overwrite.

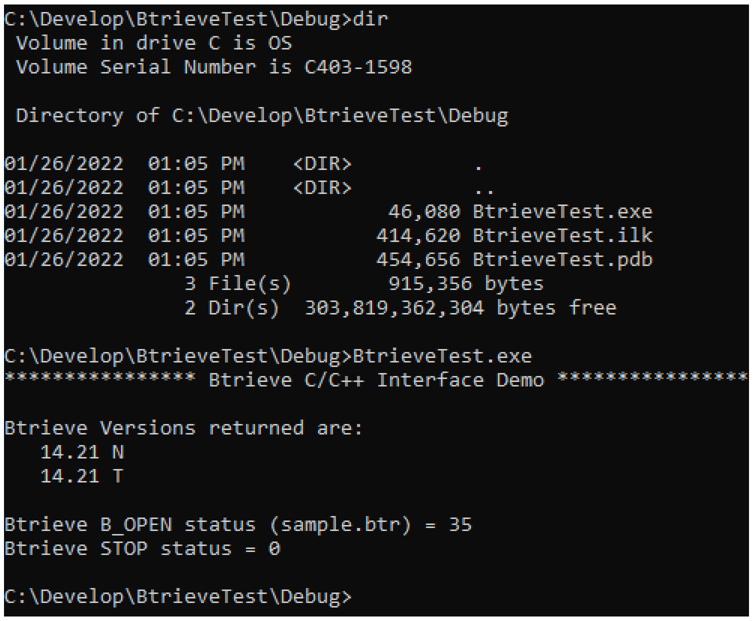
In older versions of Visual Studio, these issues simply caused warnings to appear, but newer versions of Visual Studio flag these as hard errors. There are two ways to fix it:

- You can set the compiler directive _CRT_SECURE_NO_WARNINGS (in the same place you defined BTI_WIN_32). This will disable all of these warnings and allow your code to compile. Of course, if you overwrite memory, strange results are your own fault.
- You can change all of the calls to use their secure variants. This requires changing the function name and adding the extra target length parameter. Note, though, that this can make your code less portable to a different compiler in the future.

We're going to use the Microsoft recommendation and change lines 308/309:

```
strcpy_s((BTI_CHAR*)keyBuf1, 255, FILE1_NAME);
strcpy_s((BTI_CHAR*)keyBuf2, 255, FILE2_NAME);
```

Think you're done now? Indeed you are! The project now compiles and runs!



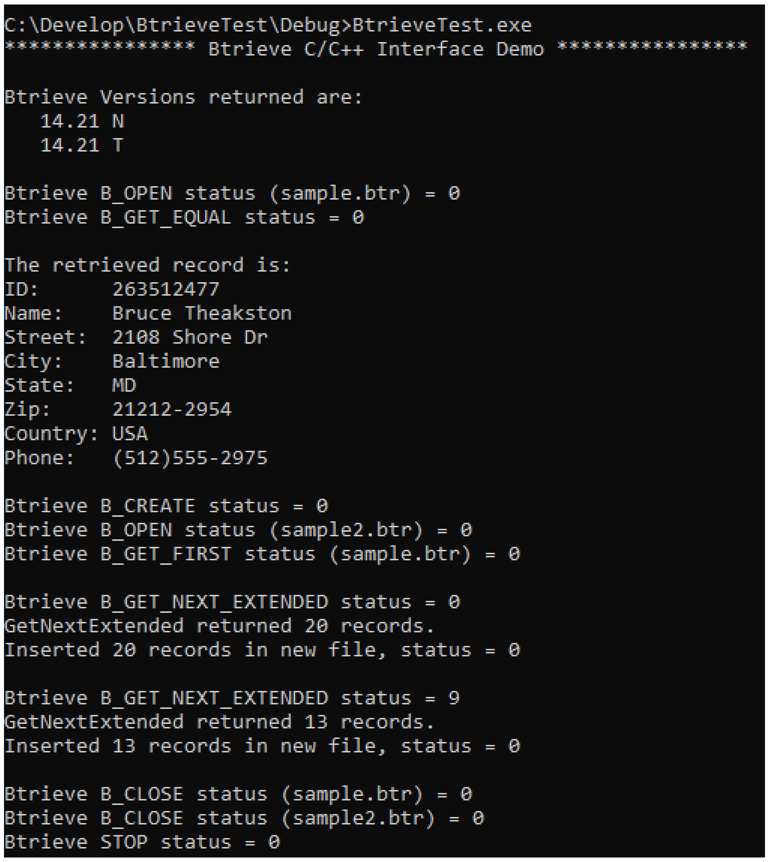
```
C:\Develop\BtrieveTest\Debug>dir
 Volume in drive C is OS
 Volume Serial Number is C403-1598

 Directory of C:\Develop\BtrieveTest\Debug

01/26/2022  01:05 PM    <DIR>          .
01/26/2022  01:05 PM    <DIR>          ..
01/26/2022  01:05 PM            46,080 BtrieveTest.exe
01/26/2022  01:05 PM           414,620 BtrieveTest.ilk
01/26/2022  01:05 PM           454,656 BtrieveTest.pdb
               3 File(s)        915,356 bytes
               2 Dir(s)  303,819,362,304 bytes free

C:\Develop\BtrieveTest\Debug>BtrieveTest.exe
**************** Btrieve C/C++ Interface Demo ****************

Btrieve Versions returned are:
   14.21 N
   14.21 T

Btrieve B_OPEN status (sample.btr) = 35
Btrieve STOP status = 0

C:\Develop\BtrieveTest\Debug>
```

However, if you look closely, you'll see that we get an error (35) when this is running. Looking up this error in the header file, we find that this equates to "`B_DIRECTORY_ERROR`". This error is returned because Actian changed the default location of the SAMPLE.BTR file between the time this sample code was created and the current v14/v15 releases.

Thankfully, fixing it is as easy as changing PSQL to Zen in two lines, 65/66:

```
#define FILE1_NAME "c:\\ProgramData\\Actian\\Zen\\samples\\sample.btr"
#define FILE2_NAME "c:\\ProgramData\\Actian\\Zen\\samples\\sample2.btr"
```

Rebuild the application and run it, and you should now get a full test run of the sample code which opens a file, reads a record and displays some data from it, then copies a set of records from one file to another. Whee!



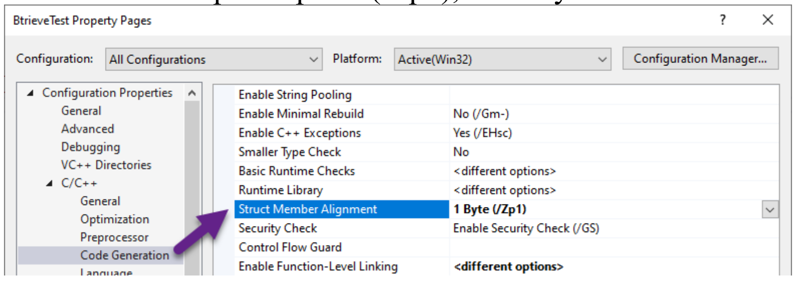
```
C:\Develop\BtrieveTest\Debug>BtrieveTest.exe
**************** Btrieve C/C++ Interface Demo ****************

Btrieve Versions returned are:
   14.21 N
   14.21 T

Btrieve B_OPEN status (sample.btr) = 0
Btrieve B_GET_EQUAL status = 0

The retrieved record is:
ID:      263512477
Name:    Bruce Theakston
Street:  2108 Shore Dr
City:    Baltimore
State:   MD
Zip:     21212-2954
Country: USA
Phone:   (512)555-2975

Btrieve B_CREATE status = 0
Btrieve B_OPEN status (sample2.btr) = 0
Btrieve B_GET_FIRST status (sample.btr) = 0

Btrieve B_GET_NEXT_EXTENDED status = 0
GetNextExtended returned 20 records.
Inserted 20 records in new file, status = 0

Btrieve B_GET_NEXT_EXTENDED status = 9
GetNextExtended returned 13 records.
Inserted 13 records in new file, status = 0

Btrieve B_CLOSE status (sample.btr) = 0
Btrieve B_CLOSE status (sample2.btr) = 0
Btrieve STOP status = 0
```

Closing Remarks

We have covered most of the more maddening issues that you're likely to run into thoughout this paper. However, I'd like to leave you with a few additional thoughts:

- If you switch between DEBUG and RELEASE modes in your application and things suddenly don't build properly, check the configuration settings. It is VERY common to modify the project settings and forget to change the Configuration box to All Configurations. You then make a required change, and when you switch to the other config, it breaks. Be very careful and always check this box!

- In order to force structure alignment on a byte boundary, it used to be common to set this as a compiler option (/Zp1), which you can do here:



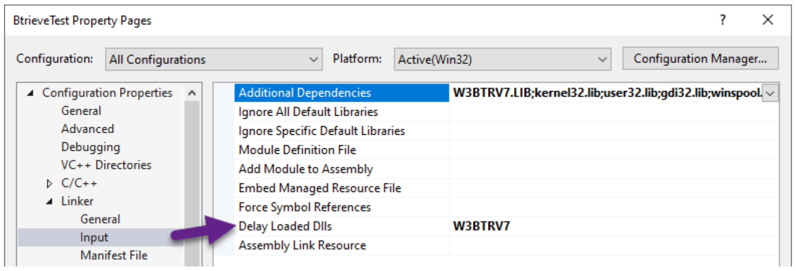
Configuration: All Configurations Platform: Active(Win32)

Enable String Pooling	
Enable Minimal Rebuild	No (/Gm-)
Enable C++ Exceptions	Yes (/EHsc)
Smaller Type Check	No
Basic Runtime Checks	<different options>
Runtime Library	<different options>
Struct Member Alignment	**1 Byte (/Zp1)**
Security Check	Enable Security Check (/GS)
Control Flow Guard	
Enable Function-Level Linking	**<different options>**

However, some newer Visual Studio libraries simply cannot be used when called with byte alignment, and the linker will spit back errors if you try this. As such, you will want to be sure to always use pragma statements to force alignment around your Btrieve structures. Better yet, you can use the pragma stack to preserve the original alignment outside of your code. This wills like this:

```
#pragma pack(push,1)
typedef struct
{
   BTI_UINT32 Value1;
   BTI_BYTE Value2;
   BTI_UINT32 Value3;
} MY_DATA_STRUCTURE;
#pragma pack(pop)
```

- If you will be building multiple projects over the years, then be sure to spend some time up front learning how to set up the default libraries, so that you can avoid copying the header and BTRAPI.C files into each project as you go along. A bit of time setting up the environment up front can save you hours of effort in the long run.
- If you use a newer SDK (from v13 and above) to build an application, then you may find that this application refuses to run on the older environments (v12 and older) due to a missing link for the BTRVEX function. This is caused by the newer SDK Library (W3BTRV7.LIB) including the function definitions for the BTRVEX and BTRVEXID functions. You can prevent the loader from resolving every symbol when the program loads by setting the **Delay Loaded Dlls** option for this file:



BtrieveTest Property Pages

Configuration: All Configurations   Platform: Active(Win32)   Configuration Manager...

Additional Dependencies	W3BTRV7.LIB;kernel32.lib;user32.lib;gdi32.lib;winspool.
Ignore All Default Libraries	
Ignore Specific Default Libraries	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded Dlls	W3BTRV7
Assembly Link Resource	

- If you are looking to use other functions, such as the Distributed Tuning Interface (DTI) which affords access to the configuration and monitoring of the database environment, you will need another SDK download (Zen-SDK-DTI-Windows-noarch-14.20.012.000.exe) which, in turn, includes the needed header and import library files. Setup is similar to that which is covered here.

Of course, if you still can't get it to work, contact Goldstar Software and let us help!