Accessing Zen v14 from PowerShell Using the ODBC Interface

A White Paper From

GOLDSTAR SOFTWARE

www.GoldstarSoftware.com

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

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 lower-level interface, commonly known as the Btrieve API, to provide the needed performance and flexibility. However, other interfaces leverage SQL through several common interfaces, including ADO and ODBC. The nice thing about the ODBC interface is that it works to a wide variety of databases using the "least common denominator" access methods. However, for simple operations, you can accomplish quite a bit through these standard calls.

When accessing any database via ODBC, you have to have a programming environment in which to write your code. With some API's, you must stick with 3G and 4G languages like C, C++, Magic, Delphi, COBOL, and so forth. However, other API's (including ODBC) are directly accessible through scripting environments like VBScript and PowerShell.

This paper provides a straightforward example of using PowerShell to perform a simple update task. Imagine the following scenario: you receive a comma-delimited file from your vendor which contains inventory part data and pricing, and you want to update the pricing directly in your database. Typing this by hand is error prone, and with 15,000 parts from this vendor, it would be time consuming to have to check every price with each new price list update. The basis for this example is taken from a real-world customer example.

Important Pre-requisites

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

1) You should have a Windows operating system properly installed with PowerShell available.
2) You should have Zen v14 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 v14 Workgroup Engine or Server Engine installed where the database files are located, and the Zen v14 Client installed on your development computer.

Clarifying the Problem Statement

The first step of any development project is to develop or refine the **problem statement** so that you know exactly what you are trying to do. You can then move on towards defining the functional requirements (what the code must do) and qualitative requirements (how the code must do it), and then you can start getting into the exact specifications (i.e. data input formats, data output formats, data structures needed, and so on).

This is a very simple project, so we will lump these all in together. First off, we have an application that has an **InventoryPart** table defined within it. This application is using Zen v14, and data dictionaries have been provided, so we have enabled SQL/ODBC access to the data through an ODBC data source. The table has many fields, but we are saved by the fact that ODBC doesn't care about every field– we only need to know the fields that we are working with, namely the **SKU** (a unique ID for each inventory part) and **Price** (a numeric value representing the price). Second, we receive an import file called "PriceFile.csv" which is in standard comma-delimited format and includes (among other fields) both part number and new price fields. Finally, the goal of this project is to read each line from the import file, parse out the SKU and Price fields, and then use that to update the table in the database accordingly.

Verifying the Source and Target Data

If we did our job correctly, then the Problem Statement, Requirements, and Specifications documents will already contain the exact file names field names, data types, and other information. However, it doesn't hurt to double check this right now. You'll then use the real field names in the scripts below.

However, if you are following along through this document as a tutorial and you want to be able to run the resulting project without modifications, then you should do the following steps to prepare for running it:

1. Use Notepad to create an import file called **PriceFile.csv** with the following lines:
   ```
   SKU,Description,Price
   Z14S006,Zen v14 Enterprise Server 6-User,1495
   Z14S010,Zen v14 Enterprise Server 10-User,2195
   Z14S020,Zen v14 Enterprise Server 20-User,3795
   ```
2. Use the Zen Control Center to create the **InventoryParts** table in the DEMODATA database using the following two statements:
   ```
   CREATE TABLE InventoryParts (
   SKU CHAR(20),
   PartDescription VARCHAR(100),
   Price INTEGER);
   CREATE UNIQUE INDEX InventorySKU AS InventoryParts(SKU);
   ```
3. Use the Zen Control Center to add these 4 records into your new **InventoryParts** table:
   ```
   Z14S006, Zen v14 Server 6-User, 0
   Z14S010, Zen v14 Server 10-User, 0
   Z14S020, Zen v14 Server 20-User, 0
   Z14S035, Zen v14 Server 35-User, 0
   ```

Creating a PowerShell Script

PowerShell "scripts" are simply text files that have the extension of ".PS1". You can use Notepad to create and work with the files, or you can use the PowerShell IDE environment. If you are new to PowerShell, then you may have some setup tasks, such as allowing unsigned PowerShell scripts to run from your account. See the Microsoft documentation for additional information about setting up the entire PS environment.

In our case, we are going to create a simple text file called "UpdatePricesFromCSV.ps1" and open it, then paste in the code from Appendix A.

Understanding the Script

Our sample code is broken down into 3 sections, so let's look at each in turn.

The first section is used to set up some values that will be used elsewhere in the script to make changes a bit easier. While we could also just use the appropriate text later on where needed, this design makes it very easy to modify the script without making some inadvertent change that breaks something.

- Line 1 provides the path to the import file, PriceFile.csv.
- Lines 2-5 provide the information needed to establish the database connection, including the server name (*localhost* is a special name that means the local computer that I am running on), the database name (Demodata if you created the sample data above, or your actual Named Database as defined on the server), and login credentials (if you have security enabled). Note that if you do NOT have security enabled, the credentials can be anything.

The second section sets up the ODBC access environment.

- Line 7 creates a new ODBC Connection string that specifies the 64-bit Zen v14 ODBC Driver and the server connection information entered from Section 1. (This driver name assumes that I am using a 64-bit PowerShell. If you are running a 32-bit operating system with a 32-bit PowerShell, then use the driver name of "Pervasive ODBC Client Interface" instead.)
- Lines 8-10 create a new ODBC Connection based on the connection string and then opens the connection. This establishes communications with the engine and the database specified.
- Lines 11-12 then create a new ODBC Command object linked with the connection we just created. This process creates an ODBC "cursor" that will be used to execute the ODBC call.
- Line 13 specifies the SQL statement to be used for the operation, which is a SQL UPDATE query. The question marks (?) in the query define placeholders for values that will be replaced in each query via parameters.
- Finally, Lines 14-15 define the two parameters that will be used by the query with placeholder values. Note that while some ODBC drivers support named parameters, the Zen environment does NOT do so. Instead, parameters are always specified in the order in which they occur in the query itself, and the names are simply placeholders to make reviewing at the code easier.

The third section is where the real work of the process is done.

- Line 17 leverages a PowerShell function that reads the entire *PriceFile.CSV* file into memory, parses each line into the fields, and then returns a collection variable called *PriceChanges*. While this line looks simple, it hides a lot of the complexity of parsing values from such a file. Here you can see a glint of the real power of PowerShell!
- Line 18 starts a loop to iterate over every row in the *PriceChanges* collection. With each iteration, the *Change* variable is updated to the collection of fields in the given row.
- Lines 19-20 set up each of the parameter values for this iteration, pulling the fields from the *Change* collection.
- Line 21 fires the UPDATE query with the parameterized values, actually modifying the data.
- Line 22 simply acts as a progress meter by just writing some text to the screen.
- Line 23 completes the loop, which will run until the program is done iterating over the entire incoming data set.
- Line 24 shuts down the database connection after we are all done.

I should also note that this script was written as a teaching tool, and it should not be considered a well-written script over all. First of all, it lacks error handling of any kind, so it will choke on many errors, like PriceFile.CSV Not Found, Server Unreachable, Database Name Incorrect, or bad login credentials. In addition, the code has been written for simplicity and clarity, whereas many of the statements could be combined or otherwise written more concisely.

Running Your PowerShell Script

If you are using the PowerShell IDE, you can run the script by simply clicking the Run button (the green arrow in the tool bar), and it should execute the SQL queries needed to update the data.

If you have created the script with Notepad, then be sure to save it as a PS1 file (not .PS1.TXT) first. Then, go to a command prompt and start PowerShell with the command "powershell" by itself. This will change your prompt to include "PS" at the beginning. You can now run your script by simply typing the script file name.

[If you get an error when attempting to run this script, it may be because you have not enabled the use of unsigned scripts. The error message is fairly clear, and it even tells you how to fix it. If you're still stumped, Google the message, and you'll find plenty of suggestions on working around it.]

To verify that it is working correctly, use the Zen Control Center to view the data records in the Demodata.InventoryParts table. You should see that the first three records should have been updated accordingly. You have now successfully written your own ODBC/PowerShell script!

Expanding Your Knowledge

Want to know if you understand what is going on? Test your understanding by trying the following:

1) Explain what happened to the "Description" field in the import file?
2) Modify the script to update BOTH the Price and the PartDescription fields in the target table based on the incoming data. (You'll need to modify the SQL statement AND add another parameter in the correct location.)
3) Add error handling on the import file so that the script doesn't crash if the source file is gone, but instead returns a user-friendly error message.
4) Add a user prompt for the import filename. (You may wish to use `Read-Host` for this.)
5) Add error detection on the SQL query, so that you can detect a situation where you don't have a corresponding row in the existing table.
6) **For expert developers only:** Try changing the query to use an UPSERT command, which attempts an UPDATE first, but then automatically falls back to an INSERT statement if the data to be updated doesn't exist. This minor alteration can reduce the error handling needed by the PowerShell script itself and allow you to handle both new and old SKUs in the same loop.

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 Script

The following script makes use of the ODBC API and can be used as instructional and sample code.

```
$PriceFileName = ".\pricefile.csv"
$DataServer = "Localhost"
$DatabaseName = "Demodata"
$UserName = "Master"
$Password = "password"

$connstring = "Driver={Pervasive ODBC Interface};Server=$DataServer;DBQ=$Databasename;UID=$UserName;PWD=$Password;"
$conn = New-Object System.Data.Odbc.OdbcConnection
$conn.ConnectionString = $connstring
$conn.open()
$cmd = New-Object System.Data.Odbc.OdbcCommand
$cmd.Connection = $conn
$cmd.CommandText = "UPDATE InventoryParts SET Price = ? WHERE SKU = ?;"
$cmd.Parameters.AddWithValue("@Price", "") | Out-Null
$cmd.Parameters.AddWithValue("@SKU", 0) | Out-Null

$PriceChanges = Import-Csv -Path $PriceFileName
foreach ($Change in $PriceChanges) {
    $cmd.Parameters[0].Value = $Change.Price
    $cmd.Parameters[1].Value = $Change.SKU
    $cmd.ExecuteNonQuery() | Out-Null
    Write-Host "SKU $($Change.SKU) Now Has a price of $($Change.Price)"
}
$conn.close()
```