

SQLExec v2.65 for Win32

Program Description	<p>SQLExec is a Win32 application that accepts SQL statements or script files and executes each statement in the ODBC environment.</p> <p>Before attempting to use SQLExec, verify that you have a valid System or User DSN created. As the method of creating a DSN is different for different databases, please see your database engine documentation for additional help.</p>
Platform and Package	Win32; Stand-Alone Utility Only; GSLic
Pricing	<p>\$200 Per Workstation License</p> <p>\$800 Per LAN Site License</p>
Command Line Syntax and Help Screen	<p>SQLEXEC Version 2.65: 03/01 (C)2023 Goldstar Software Inc. Registered to Goldstar Software Inc. (Site License GS)</p> <p>Usage: SQLEXEC <DSN> ["Statement"] [/options] This command executes SQL Statements Against a DSN. <DSN> is the ODBC Data Source Name with which to connect. "Statement" Is the optional SQL statement to execute (up to 16K chars). /B Enables Blank Processing (Eliminates Trailing Spaces From Data). /C<filename> to read a connection string from a file instead. /D Enables Debugging Output, or /DT Enables Timestamp Output. /H disables the column Header row (or ignores with /I). /I<filename> uses a data import file (SDF) in an INSERT statement. /K Keeps comments and other formatting from SQL statements (for Procs). /L# loops to run the statement # times, useful for performance testing. /M<filename> executes multiple statements from the given filename. /Ox changes Output format (F=Fixed, T=Tab-Separated, B=Bar-Separated). (C=CommaQuote, R=Raw Comma, S=FS/RS-Delim, H=HTML, X=XML, J=JSON). /P<password> specifies an ODBC Password for this User Name. (*=Entry) /QT# sets a timeout (in secvonds) on the query (Default 0, No Timeout) /R repeats the SQL statements to the output stream. /S<symbol> specifies the separator symbol in the input file. (Default=#) /T disables ODBC AutoCommit Option for script. /U<username> specifies an ODBC User Name to use (Default=Master). /W# indicates ms to wait between SQL statements (Default 0). /Z# indicates how to handle end-of-file (0=CRLF, 1=Add EOF, 2=NoCRLF). Field Filtering Options: (Add multiple option values together, if needed.) /F1 Change CR/LF to space character. /F2 Change Null byte (0) to space character. /F4 Change control characters (1-31) to space character. /F8 Clear high bit on every character. /F16 Force all text characters to UPPERCASE. /F32 Change CR/LF Pairs to "\n". Quote Filtering Options: (Use /QS# to affect Single quotes) /QD1 Remove all Double Quote characters. /QD2 Double all Double Quote characters (""). /QD3 Change all Double Quote characters to underscore (_). /QD4 Convert all Double Quote characters to ASCII (&#34;). /QD5 Convert all Double Quote characters to WEB (%34). /QD6 Convert all Double Quote characters to OCTAL (\042). /QD7 Convert all Double Quote characters to HEX (\x22). /QD8 Convert all Double Quote characters to UNICODE (\u0022). /QD9 Escapes all Double Quote characters with Backslash (e.g. \"). Error Logging Options: /EO Logs Errors to StdOut. (Default is logging to stderr.) /EL[file] Logs to SQLEXEC.ERR or optional text file provided. /EC[file] Logs to SQLEXEC.ERR [or file] as Comma-Separated Values. (Single-letter options can be combined.)</p> <p>To run a single statement from the command line: SQLEXEC DEMODATA "Select * from Person"</p> <p>To run a script with multiple statements from a file called SCRIPT.SQL: SQLEXEC DEMODATA /MScript.SQL</p> <p>Input Required:</p>
Examples and Sample Usage	

The /U and /P options can be provided to log into the database, if security is enabled. You can optionally use a connection string from a separate text file that has all of these settings defined in it, if needed. In this case, be sure to specify a DSN (like “none”) as a placeholder for this required parameter. If you use an asterisk (*) for the password (i.e. /P*), then the user will be prompted for a secured password when running the query.

If the Connection String file is used, the file should be a simple text file with one line only – the ODBC Connection String for the data source. See Microsoft Developer documentation for full details about this string.

The /M option allows you to run multiple statements with the same function call, reading a script from an external text file. The script file should separate multiple statements with a pound sign (#) or a user-definable character (the /S option). If you want the statements echoed to the output stream, add the /R switch.

By default, SQLExec uses a standard ODBC connection, where each command is automatically committed. If you are writing a script using transactions and you wish to force your own COMMIT operations, then you can add the /T option on the command line, and auto-commit will be disabled.

Normally, SQLExec strips out comments, tabs, returns, duplicate spaces, and other such text before submitting a SQL query to the engine. If you are running a script in which you want to keep all of the script formatting, use the /K option. This can be especially useful if you are creating stored procedures that include comments.

Output Returned:

The default output of this program will be a series of rows displayed as a comma-delimited list of all fields requested from the SQL statement (if a SELECT statement) or nothing (in the case of an INSERT or UPDATE statement). The first row will be a header row defining each field in the query. With this output, you can easily import the data into MSAccess or MSEXcel, or other applications.

To eliminate the header row (useful if you are concatenating the output from several smaller Subqueries), add the /H switch to the command line.

Normally, the fields are displayed as space-padded fields to their full length. Use the /B switch to eliminate trailing blanks from any fields.

If you want the output to go to a file, you can either redirect stdout to a file (through the redirect character “>” on the command line), or you can include a target filename inside dollar-sign delimiters at the beginning of the statement, as in this example: “\$filename.out\$SELECT ...”. To append the output to a file, either use “>>” on the command line to redirect stdout, or include the target filename in ampersand delimiters: “&filename.out&SELECT ...”.

Output Formatting:

If you are displaying data on-screen, the default comma/quote-delimited output (/OC) can get a bit difficult to read. The /OF switch will output the data as fixed-width fields instead, filling the extra space with SPACE characters. The width of each field will be the data field width or the column heading width, whichever is larger. Of course, this export format can also be useful for direct-to-text reporting, or even direct-to-printer reporting!

If you don’t want the delimiter quotes in the output, you can export the data as raw comma-delimited with the /OR switch. This outputs each field with a single

comma as the separator, with no quotation marks. If you are exporting data that contains quotation marks or commas, however, then these elements may cause your application to import data incorrectly. In this case, try using /OT (tab delimited) or /OB (vertical-bar delimited) options instead.

The /OH switch will export data to an HTML formatted file. Be sure to redirect this to a file for storage, and then display it from the browser. An enhancement has been added for improved CGI scripting support.

The /OX switch will export data to an XML formatted file. Be sure to redirect this output to a file for storage.

The /OJ option will export data to a JSON-formatted file, suitable for use with a web front-end or any other environment capable of parsing the JSON stream.

Here is a sample output:

```
R:\>SQLExec DEMODATA "SELECT TOP 1 * FROM Billing" /oj /b /r
SQLEXEC Version 2.63: 04/28 (C)2021 Goldstar Software Inc.
Registered to Goldstar Software Inc. (Site License GS)
{
  "Description":"SQLExec",
  "Title":"SQLExec Query Results",
  "VersionString":"SQLEXEC Version 2.63: 04/28 (C)2021",
  "Statements": [
    {
      "Statement":"Select top 1 * from Billing",
      "Results": [
        {
          "Student_ID":"100062607",
          "Transaction_Number":"1",
          "Log":"1996-03-28 16:38:52.000",
          "Amount_Owed":"2125.00",
          "Amount_Paid":"1031.25",
          "Registrar_ID":"130312616",
          "Comments":""
        }
      ],
      "StatementStatus":"0"
    }
  ],
  "SQLExecStatus":"0"
}
```

Statements is an array of returned structures, each containing a Statement and Results elements. If multiple statements are sent to SQLExec using the /M option, then Statements will be a multi-element array as well.

For exporting to the mainframe-compatible Field-Separated format, use the /OS option. This option changes the export format to separator-delimited, using the Field Separator character (FS, or ASCII 28) between fields and the Record Separator Character (RS, or ASCII 30) between each record. Some mainframe applications will handle these codes natively, while other applications may need to be adjusted to work with this format.

SQLExec normally exports fixed-length and other data with CR/LF pairs at the end of each line. Some import utilities require an end-of-file marker (Ctrl-Z) after the last record, and you can add this extra byte by including the /Z1 switch. Other import utilities choke if a final (trailing) CR/LF pair is included, with no data coming after it. In this case, use the /Z2 switch, and the trailing CR/LF pair will be omitted from the output.

Running Insert Statements:

Importing data is also a key feature of SQLExec. If you put an INSERT statement onto the command line, which can be as simple as just "INSERT INTO TABLE" or as complex as having an entire field list (up to 1000 characters), then

specify the /I option and provide a comma-delimited file, then SQLExec will read each line out of the file, build a VALUES clause, and import the data into your database for you.

A few caveats exist:

- ❑ Currently, the INSERT statement MUST be on the command line. There is no support for super-long INSERT statements.
- ❑ The import data is limited at 64K per line. The longest “literal” string allowed in the SRDE is 14997 characters (15000 with the quotes and the NULL terminator). This may be addressed in the future (using SQLPutData statements) if demand exists for this feature.
- ❑ The file must be in a comma-delimited format, with a carriage return marking the end of each row. Embedded carriage returns are not allowed as they are for statement files.

The /H option will skip the first row of the import data file, which is useful if it contains a header row that you don’t want to have to remove manually.

If your import data file contains blank fields, you can use the /B option to convert the empty fields to NULL values prior to submitting the statement.

Field Filtering:

Some database applications store an extensive amount of text data and allow you to store control characters, including carriage returns, line feeds, or NULL bytes, inside the actual data. Other databases will have improperly-defined fields, whereby numeric or other data is included within CHAR fields. In either case, SQLExec can convert these special characters for you during the export.

When it sees the /F option, SQLExec interprets the numeric value following it to select one or more filters on EACH data field (there is currently no support for single-field filtering). The currently-defined filters include:

- 0: No Filtering.
- 1: Convert all CR/LF characters to a space character.
- 2: Convert all NULL bytes (ASCII 0) to a space character.
- 4: Change all control characters, i.e. those in the range from 1 through 31, to space characters. (This currently includes CR/LF bytes as well.)
- 8: Clear the high bit on each byte. This is useful for databases that work with the high bit set, while the target database wants low-order ASCII values.
- 16: Force all alphabetic characters to uppercase. This can be useful if source data is a mixture of upper and lower case, and you need the target data in a standardized (all uppercase) format.
- 32: Convert any CR/LF pairs to “\n”. This can be useful for maintaining formatting of a large text field, as the “\n” can be easily converted to a CR/LF in the final output.

You can combine multiple filtering options, if needed, by adding the values together (they are simple bit flags). For example, /F20 will enable the UPPERCASE filter and also change all control codes to spaces.

Quote Filtering:

Some database applications also store included double quotes (") or single quotes

() inside the data itself. These special characters can play havoc with many import routines. To counter this, there are several Quote Filtering options available. To filter out double quotes, specify /QD followed by a number of the option you want. To filter single quotes, use /QS followed by a number.

The currently-defined quote filters include:

- 0: No Filtering.
- 1: Remove any quote characters found.
- 2: Double any quote characters (e.g. from " to "" and ' to ").
- 3: Convert quote characters to underscores (_).
- 4: Change quote characters to the ASCII format (e.g. """), commonly used by some environments like MySQL.
- 5: Convert quote characters to the web format (e.g. %34) for use with browsers.
- 6: Convert the quote character to its octal value (e.g. \042).
- 7: Convert the quote character to its hex value (e.g. \x22).
- 8: Convert the quote character to Unicode (e.g. \u0022).
- 9: Escape the quote characters with the backslash character, as in \".

Error Handling:

Normally, errors are logged to stderr, allowing you to redirect stdout (1>) to a file for your output, and stderr (2>) to get the error messages separately. If you want them included in the same output file, use the option /EO to send ODBC error messages to stdout. To use a separate text log file, use the /EL option. BY default, this places all errors on their own line in the file called SQLEXEC.ERR. If you wish to use a different file (to keep track of different process errors separately), then add the error log filename after the /EL, as in /ELERROR.LOG. If you will be examining the errors with a database application, then use the /EC option to output all errors to the comma-separated value file SQLEXEC.ERR, or provide your own filename as above.

When the program exits, it will set the environment variable ERRORLEVEL that can be queried to determine the exit status. The following values are currently defined:

- 1 = Bad Arguments On Command Line
- 2 = Beta Period Has Expired
- 3 = Unable to Read Connect String File
- 4 = Unable to Connect to Specified Database
- 5 = Error while Executing Statement.
- 6 = Error Shutting Down.

Note that errors caused by bad data are NOT a cause for exiting the application, but instead are simply displayed as errors.

Query Performance Testing:

You can use SQLExec for query performance testing, too. The /L switch will allow you to specify the number of times to run the given statement. (If you do not specify a number, the default is 1.) You can use this feature, along with the /DT (Display Times) option, to run a statement a large number of times (say 10,000) and compute the total execution time. Looping like this will magnify small

differences in query performance, making it easier to see how two statements differ in overall system workload and performance. If you need to delay a script file for some reason (like monitoring a multi-statement script), then you can use the /W option to insert a separate delay after each SQL statement is run.

Other Information

SQLExec is a standalone utility from Goldstar Software.

While written for Actian PSQL/Zen, this utility has been successfully tested against SQLServer, Oracle, and Sybase databases and found to be functional using both DSN's and Connection Strings.

For more information on our utilities, contact us at www.goldstarsoftware.com

Version History

Version 1.0: First documented version

Version 1.1: Added support for /C (Connection String).

Version 1.2: Added switch to disable column header row.

Version 1.3: Improved runtime error handling of some SQLState conditions. Added /W switch to display fixed-width output.

Version 1.4: Standardized return code values for ERRORLEVEL and added Error Logging Options.

Version 1.5: Fixed output bug that duplicated the CR code.

Version 1.6: Changed /W switch to multi-format /O switch and added ability to export data as separator-delimited in addition to comma and fixed-width. Changed /F(file) switch to /M to make room for Filter switch. Added /F filter switch to better handle "garbage" in the data streams. Expanded maximum field length from 80 to 65530 characters (ODBC maximum). Expanded maximum command-line statements length from 253 to 16K (same length as file-based statements).

Version 1.7: Added Tab-Delimited, Bar-Delimited, and Raw Comma-Delimited output format options.

Version 1.8: Added ability to use INSERT statements to Import data (/I) directly from an SDF file.

Version 1.81: Fixed bug in handling the reading of data with invalid date/time fields. This program was exporting data from the previous record – it now exports null fields after the invalid field, and it dumps a debugging log so you can see which field failed.

Version 1.90: Added ability to export data to HTML format as well.

Version 1.91: Fixed bug with capitalization of username/password.

Version 1.92: Added /R option to echo SQL statements to the output stream.

Version 1.93: Increased maximum INSERT statement size to 65000 bytes.

Version 1.94: Added ability to filter double quotes and single from the data stream on the extraction.

Version 1.95: Fixed issues with importing data containing single and double quotes.

Version 1.96: Eliminated blank line from output when no header row displayed. Send failed SQL statement to stderr when errors occur in ExecDirect.

Version 2.0: Added GSLicense functionality.

Version 2.01: Improved processing of some error messages.

Version 2.10: Added Debug Timestamp option (/DT) for profiling queries.

Version 2.20: Added ability to include target filenames in queries.

Version 2.21: Added ability to append to existing target filenames.

Version 2.22: Skip any statements less than 3 characters. This allows user to start the multi-statement script with a pound sign.

Version 2.23: Added filter to remove comments from the source file before submitting the statements.

Version 2.30: Added quote filtering options and deprecated /F32 and /F64 options.

Version 2.40: Added ability for user to enter password at runtime with the /P* option. Password is hidden from view, but length is visible for ease of use.

Version 2.41: Added ability to convert CR/LF pairs to \n.

Version 2.50: Added ability to export data in XML format.

Version 2.51: Fixed an issue with the field width in fixed-length output format. If the header row is enabled, the width will be the maximum of either the column or the header text. If the header row is disabled, the width will be the column width only.

Version 2.52: Fixed an issue with the /OF output format and exporting to a file.

Version 2.53: Added the /Z switch functionality.

Version 2.54: Addressed an issue with “//” comments, and enabled a new feature to preserve comments (/K) in scripts.

Version 2.55: Removed feature that stripped out formatting of SQL statements to save bytes on long queries – it was not worth the hassle. If your query is over 32K, you may need to edit it manually.

Version 2.56: Removed extraneous LF character on output.

Version 2.57: Added delta time calculations for Query Execution and Data Read.

Version 2.58: Updated licensing code.

Version 2.60: Added /L switch to loop over a statement a large number of times.

Version 2.61: Added handling of /B and /H switches on INSERT statements.

Version 2.62: Added the /W (wait) switch for script debugging, as well as the /T switch to disable transaction auto-commit.

Version 2.63: Added the JSON output format.

Version 2.65: Added option to specify the Query Timeout (/QT) for testing.

Known Problems

The handling of invalid date/time fields is sometimes flaky. The system should be able to recover from this issue and export the remaining fields, but it just stops. Debugging info had been added, but some substantial time is needed to troubleshoot this issue.

It is not currently possible to access a database with the password of a single asterisk (*).