


[print](#) | [close](#)

APIs by Example: Reverse Engineering Database Files and Objects to SQL DDL Statements

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 08/09/2007 (All day)

We've been covering the topic of using SQL instead of DDS to define your database files for quite some time now. The benefits and possible problems and pitfalls have been documented in a number of articles, as have the various methods and approaches to actually making the move from DDS to SQL. To get you up to speed in case you missed any of the articles when they were initially published, I've included links at the end of this article. Please note that some of the articles require a ProVIP membership to access.

Reading these articles, you'll notice that there's an option of using iSeries Navigator to take existing DDS-defined database files and generate the SQL Data Definition Language (DDL) statements needed to create or re-create these objects as SQL file objects, such as tables, views, and indexes. This iSeries Navigator facility is based on an i5/OS API: Generate Data Definition Language (QSQGNDDL). And although iSeries Navigator continues to improve in performance and usability with every new release, in some cases using a CL command to get the job done is still faster and more convenient. With that reality in mind, I took advantage of the QSQGNDDL API to create a Generate SQL Data Definition Language (GENSQLDDL) command that offers the same reverse engineering services that iSeries Navigator offers.

The QSQGNDDL API — and consequently the GENSQLDDL command — is not limited to generating DDL based on DDS-defined file objects; it can take any SQL database object and create the SQL statements required to create or re-create the object in question. The following database object types are supported:

- TABLE or PF — An SQL table or physical file
- VIEW or LF — An SQL view or logical file
- SCHEMA or LIB — An SQL schema (collection) or library
- ALIAS — An SQL alias
- INDEX — An SQL index
- TRIGGER — The object attribute is a trigger
- CONSTRAINT — The object attribute is a constraint
- FUNCTION — An SQL function
- PROCEDURE — An SQL procedure
- TYPE — The object is an SQL type

The GENSQLDDL command also has many formatting parameters that let you control which SQL statements are included in the DDL generation and how they are formatted. I explain a couple of the more important options in the following paragraphs.

- Naming: This option defines the naming convention used for qualified names in the generated SQL statements. The possible values are:

- *SQL — The collection.table syntax
- *SYS — The library/file syntax
- Standards option: The standards option specifies whether the generated SQL statements should contain DB2 for i5/OS extensions to the DB2 Universal Database family, SQL, or the ANSI and ISO SQL standards. The possible values are:
 - *DB2EXT — DB2 for i5/OS extensions can be generated in SQL statements.
 - *DB2STD — The generated SQL statements must conform to SQL statements common to the DB2 Universal Database family.
 - *ISOANSI — The generated SQL statements must conform to the following ISO and ANSI SQL standards: *ISO 9075-1: 1999, Database Language SQL* and *ANSI X3.135-1-1999, Database Language SQL*.

The standards option is important because it affects the type of SQL statements that the GENSQLDDL command allows in the generated source and consequently in which environments they can be successfully run. Here's an excerpt from the QSQGNDDL API documentation explaining some of the considerations involved in setting this parameter:

If *DB2STD or *ISOANSI is chosen, the SQL statements generated may not completely represent the object in DB2 UDB for iSeries; however, the statements will be compatible with the specified DB2 Family or ANSI and ISO standards option.

If the object is an SQL function, SQL procedure, SQL trigger, or SQL view, the SQL statements in the body of the object are included in the generated SQL statement. Hence, if the Standards option *DB2STD or *ISOANSI is chosen, the generated SQL statement may not conform to the specified standards option since the statements within the body of the SQL object may not conform to the specified standard. For example, if a CREATE INDEX statement exists in the body of an SQL procedure, the generated CREATE PROCEDURE statement will contain the CREATE INDEX statement even if Standards option *DB2STD or *ISOANSI is chosen.

There is no attempt to take product-specific limits into account. For example, a table name in DB2 for i5/OS can be 128 bytes, but other products might not support table names that long. Thus, even if the generated SQL statement is standard, it still might not work on other products if they have smaller limits than those on DB2.

If *DB2STD is specified:

- the naming option must be *SQL.
- the date format must be *ISO, *USA, *EUR, or *JIS.
- the time format must be *ISO, *USA, *EUR, or *JIS.
- the decimal point must be a period.

If *ISOANSI is specified:

- the naming option must be *SQL.
- the date format must be *ISO.
- the time format must be *ISO.
- the decimal point must be a period.
- an ALIAS object type must not be specified.

A comprehensive explanation of all parameters is in the command help text panel group. Here's the GENSQLDDL command prompt in its entirety:

Generate SQL Data Def Language (GENSQLDDL)

Type choices, press Enter.

Database object		
Database object library	*LIBL	Character value,
*LIBL...		
Database object type	*PF	*PF, *LF, *LIB,
*TABLE...		
Source file		Name
Library	*LIBL	Name, *LIBL,
*CURLIB		
Source member	*FIRST	Name, *FIRST,
*LAST		
Member option	*APPEND	*APPEND, *REPLACE
Statement formatting option . .	*NONE	*NONE, *FMTCHR
Naming	*SQL	*SQL, *SYS
Standards option	*DB2EXT	*DB2EXT, *DB2STD,
*ISOANSI		
Date format	*ISO	*ISO, *EUR, *JIS,
*USA...		
Date separator	'/'	/, -, ., ,, *BLANK
Time format	*ISO	*ISO, *EUR, *JIS,
*USA, *HMS		
Time separator	':'	:, ., ,, *BLANK
Decimal point	'.'	., ,
Generation severity level . . .	10	0-39
Message severity level	0	0-39
DROP option	*NOGENDROP	*NOGENDROP,
*GENDROP		
COMMENT ON statement option . .	*NOGENCOMMENT	
LABEL ON statement option . . .	*NOGENLABEL	*NOGENLABEL,
*GENLABEL		
Header generation option	*NOGENHDR	*NOGENHDR, *GENHDR
Trigger generation option . . .	*GENTRG	*NOGENTRG, *GENTRG
Constraints generation option .	*GENCST	*NOGENCST, *GENCST
System name option	*NOGENRENAME	*NOGENRENAME,
*GENRENAME		

The GENSQLDDL command's last three parameters — *Trigger generation option*, *Constraints generation option*, and *System name option* — are all V5R4 inventions. In earlier releases, the CPP therefore ignores those three parameters. The QSQGNDDL API was introduced in V5R1, limiting its use to that and later releases.

The GENSQLDDL command places the generated SQL statements in the specified source member and source file. Note that the source file must have a record length of minimum 92 bytes, otherwise the API or command fails. To execute the SQL statements, you can use the source member as direct input to the RUNSQLSTM command:

Run SQL Statements (RUNSQLSTM)		
Type choices, press Enter.		
Source file		Name
Library	*LIBL	Name, *LIBL,
*CURLIB		
Source member		Name
Commitment control	*CHG	*CHG, *UR, *CS,
*ALL, *RS...		
Naming	*SYS	*SYS, *SQL
Additional Parameters		
Severity level	10	0-40
Date format	*JOB	*JOB, *USA, *ISO,
*EUR...		
Date separator character	*JOB	*JOB, /, ., ,, -,
' ', *BLANK		
Time format	*HMS	*HMS, *USA, *ISO,
*EUR, *JIS		
Time separator character	*JOB	*JOB, :, ., ,, '
' ', *BLANK		
Default collection	*NONE	Name, *NONE
IBM SQL flagging	*NOFLAG	*NOFLAG, *FLAG
ANS flagging	*NONE	*NONE, *ANS
Decimal Point	*JOB	*JOB, *SYSVAL,
*PERIOD...		
Sort sequence	*JOB	Name, *JOB,
*LANGIDUNQ...		
Library		Name, *LIBL,
*CURLIB		
Language id	*JOB	*JOB, *JOBRUN...
Print file	QSYSPRT	Name

```

Library . . . . . *LIBL      Name, *LIBL,
*CURLIB
Statement processing . . . . . *RUN      *RUN, *SYN

Allow copy of data . . . . . *OPTIMIZE  *OPTIMIZE, *YES,
*NO
Allow blocking . . . . . *ALLREAD      *ALLREAD, *NONE,
*READ
SQL rules . . . . . *DB2      *DB2, *STD

Decimal result options:

Maximum precision . . . . . 31      31, 63

Maximum scale . . . . . 31      0-63

Minimum divide scale . . . . . 0      0-9

Listing output . . . . . *NONE      *NONE, *PRINT

Target release . . . . . *CURRENT      *CURRENT, VxRxMx

Debugging view . . . . . *NONE      *NONE, *SOURCE,
*STMT, *LIST
Close SQL cursor . . . . . *ENDACTGRP  *ENDACTGRP,
*ENDMOD
Delay PREPARE . . . . . *NO      *NO, *YES

User profile . . . . . *NAMING      *NAMING, *USER,
*OWNER

```

Note that some of the parameters are identical to those that the GENSQLDDL command offers, and they are required to match in order for the RUNSQLSTM command to run successfully.

After you install the GENSQLDDL command, you can give it a test drive by following these instructions:

1. Run the GENSQLDDL command against a file or database object of your choice — for example:

```

GENSQLDDL DBOBJ(QADBXREF)
          DBLIB(QSYS)
          TYPE(*PF)
          SRCFILE(QGPL/QDDL SRC)
          MBR(QADBXREF)
          MBROPT(*REPLACE)
          NAMING(*SQL)

```

2. Be sure to carefully read the messages and information issued during the conversion process and added to the source member. Next, adapt the generated source member to allow the generated SQL statements to be executed in the next step. You would, for example, need to

change the library in the CREATE TABLE statement to something more appropriate than QSYS, in case you have run the preceding GENSQLDDL command example.

3. Execute the RUNSQLSTM command against the newly created source member:

```
RUNSQLSTM SRCFILE (QGPL/QDDL SRC)
          SRCMBR (QADB XREF)
          NAMING (*SQL)
```

4. Verify that the file specified in the CREATE TABLE statement was successfully created. If errors occur during the execution of the SQL statement, you receive an SQL9010 exception message: *RUNSQLSTM command failed*. You can find diagnostic messages in your job's job log to point you to the actual cause of the failure.

After you complete the test, consider deleting the newly created file if you have no other use for it.

This APIs by Example includes the following sources:

```
CBX176  -- RPGLE  -- Generate SQL Data Definition Statements -- CPP
CBX176H -- PNLGRP -- Generate SQL Data Definition Statements -- Help

CBX176X -- CMD    -- Generate SQL Data Definition Statements

CBX176M -- CLP    -- Generate SQL Data Definition Statements -- Build
command
```

To create all these objects, compile and run CBX176M. Compilation instructions are in the source headers, as usual.

SQL/DDS-related articles previously published on *SystemiNetwork.com*:

[Stop Using DDS! A Better Way to Make Files \(April 12, 2007, article ID 54441\)](#)

[Follow Up to: Stop Using DDS! A Better Way to Make Files \(April 26, 2007, article ID 54545\)](#)

[Replacing a DDS Physical File with an SQL Table \(May 2005, article ID 20057\)](#)

[Performance Comparison of DDS-Defined Files and SQL-Defined Files \(May 2005, article ID 20067\)](#)

[Database Harmony: "Traditional" and SQL Coexistence \(May 2005, article ID 20060\)](#)

[Is DDS Dead? \(April 2001, article ID 9821\)](#)

IBM SQL/DDL-related documentation:

Data Definition Language:

<http://publib.boulder.ibm.com/infocenter/iseres/v5r4/topic/sqlp/rbafysqltech.htm>

Types of SQL Statements:

<http://publib.boulder.ibm.com/infocenter/iseres/v5r4/topic/sqlp/rbafystmttype.htm>

SQL Concepts:

<http://publib.boulder.ibm.com/infocenter/iseres/v5r4/topic/sqlp/rbafysqlconcepts.htm>

SQL Objects:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/sqlp/rbafysqlobjects.htm>

Run SQL Statements (RUNSQLSTM) Command:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/cl/runsqlstm.htm>

This article demonstrates the following database and file API:

Generate Data Definition Language (QSQGNDDL) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/apis/qsqgnddl.htm>

You can retrieve the source code for this API example from:

http://www.pentontech.com/IBMContent/Documents/article/55321_273_GenSqlDdl.zip.

Source URL: <http://iprodeveloper.com/rpg-programming/apis-example-reverse-engineering-database-files-and-objects-sql-ddl-statements>