SQLC – SQL Control for the Orbiter Space Flight Simulator

Pablo Edronkin, 2013 - 2019

https://orcid.org/0000-0001-8690-7030

Abstract

SQLC is a multi functional display (MFD) designed as an interface between the user and the artificial engine of the DGIIIAI spacecraft.

DGIIIAI artificial intelligence engine is designed around a relational database that uses SQL as means both to control the database as well as to execute the rules of its expert system.

A significant number of those rules are enforced automatically depending on the conditions that the ship encounters during a flight session, but since everything is expressed in terms of SQL snippets, by sending SQL queries to the expert system it is possible to influence its behavior.

That is what SQL was designed for, and it can do so using pure SQL snippets, mode and program activation and deactivation instructions as well as passing parameters directly to the knowledge base.

Keywords

AI, simulation, artificial intelligence, expert systems, databases

Acknowledgements

This project could not have taken place without the work done by:

- Dr. Martin Schweiger for developing Orbiter.
- The developers of Sqlite3.
- The Orbiter community of users and developers.

License and info for contributors

Please read the following files included with this project:

- README.md: contains info on setting up your system and installing the files related to this module.
- CONTRIBUTING.md: if you wan to contribute to this project.
- COPYING: for license information.
- Also note that this MFD would work only in conjunction with DGIIIAI or a ship designed around a compatible AI system. It is not intended for use with any other sort of OSFS ship.

Table of Contents

1. Introduction

SQLC is an MFD module made specifically to serve as an interface between the OSFS user and the onboard expert system and artificial intelligence engine of the DGIIIAI ship.

It uses a limited number of buttons that allow to connect and disconnect to a database, and operate with the expert system embedded into the code of DGIIIAI.

This expert system operates essentially based on facts characterized by a fact name or *Item* field, and a *Value*. Each fact is contained in a record of a database table called *sde_facts*.

Operations take place of the *Value* field of each record based on data that the Orbiter API sends to the system, MFD or module input, and the set of rules that the expert system has. Those rules are contained per operation in a table called *sde_rules*.

The artificial intelligence engine or AIE cyclically reads the contents of each table of the knowledge base (a name given to a database that contains expert knowledge of some type, in this case, how to fly a spaceship), the external modules such as this MFD, and applies the appropriate rules, according to the data it has at a given moment.

Then, once the rules are applied and the contents of the Value field of each fact record have been transacted, those values are returned to the OAPI and the ship does what is being told to do like increase thrust, or angle off attack, change its heading, etc.

This, in essence, is the context in which SQLC operates. For more information on how the AIE operates and the characteristics of the KB of DGIIIAI, please read **[4.]** and **[5.]** in this document and review the DGIIIAI.pdf document that accompanies the installation files for the DGIIIAI project.

2. Conventions, caveats and base conditions of development

For the SQLC project, all considerations regarding these points are the same that are applicable for the DGIIIAI project.

Terms and abbreviations used in this document are also defined in section [2.] of DGIIIAI.pdf. This document is available amongst the installation files for project DGIIIAI.

3. Development

SQLC provides a basic interface to DHIIIAI's AIE and there is still plenty of room for improvement. At the time of this writing some aspects of this project can still be developed, like:

3.1. Use of SQLC with any RDBMS.

Orbiter ships never made use of database systems. In fact, storage facilities and procedures have always been limited to flat files, which is fine for low - volume reads and writes. But AI and other complex systems might require more advanced data management.

Considering the volume of data that an OSFS session might handle, relational databases could be well - suited for almost any need while providing safe storage and data management.

There are many RDBMS systems, and some of those are free, including Sqlite3, PostgreSQL, MySQL and MariaDB. These all use SQL with varying degrees of compliance with standards, and each one of them has advantages and disadvantages. This topic is fully discussed in DGIIIAI.pdf but suffice to say for now that Sqlite3 was the choice for this project after evaluating the systems mentioned in this paragraph.

This does note mean that those other RDBMS are useless for Orbiter or the DGIIIAI project; quite on the contrary, it would be interesting to develop a layer of compatibility with those systems and DGIIIAI + SQLC.

But if you are now thinking about ODBC to achieve such goals, I would recommend against following that route due to the need for speed. It would be much better to develop C or C++ based interfaces.

3.2. Use of SQLC with other SQL compatible MFD modules.

It is perfectly possible to develop new MFD modules to deal with things like cargo, environmental

systems, etc. outside of the DGIIIAI AIE realm, and share data thorough the same RDBMS used by SQLC.

This system proves that it is possible to use a relational database in conjunction with OSFS modules; now the sky is the limit as to what can be achieved, and the best part is that using SQL any table of any database used in such a fashion can be lined to any other using just SQL queries, with no need for other interface methods.

3.3. Additional AIE functions for SQLC.

So far SQLC provides basic access and functionality to interact with DGIIIAI. There is nothing that forbids putting more specialized function on the current MFD.

4. Expert system

SQLC was designed to work as an onboard interface to DGIIAI's expert system; it does not work as an independent MFD.

It is not the scope of this document to describe the characteristics of the AIE of the intelligent Delta Glider, so for details on that matter you should read the file ~/docs/DGIIIAI/DGIIIAI.pdf and it is recommendable that you become at least acquainted with that document before attempting to use this module.

What SQLC allows the user to do is to send commands to the AIE in different ways like raw SQL instructions, specialized calls to %mode% facts, program loading and unloading and calls to facts using the fact Item and Value fields.

In this way, it should be regarded as the user interface for the AIE of the DGIIIAI project, being essentially an interface to Sqlite3 databases and specifically, those used on DGIIIAI.

You can also access the databases used by DGIIIAI and SQLC by means of any Sqlite3 compatible database manager or editor, but SQLC provides some convenience functions to do so from within an OSFS session.

4.1. AIE loop.

The AIE works on a loop composed by various steps identified by the *Status* string that each fact record is given. SQLC exchanges data only during one of these steps in order to maintain synchronization and avoid race conditions within the expert system.

The Status field is changed for all facts in *sde_facts* each time a number of tasks are finished.

4.1.1. *Status = 'sentodb'*

Data is gathered from the OAPI and facts in *sde_facts* are updated.

4.1.2. *Status = 'getfromnetwork'*

Data is exchanged with SQLC and any other external module. SQL queries should be written so that Status = '*getfromnetwork*' is a necessary condition within the statement.

4.1.3. *Status = 'applykbrules'*

Rules are applied and facts are updated based on those rules. At this point the ES evaluates each rule in *sde_rules* against the data in *sde_facts*.

4.1.4. *Status = 'dbtoact'*

Data from *sde_facts* is sent back to the ship via the OAPI and to the actuator functions that will, for example, make the ship turn, ascend, descend, retract or extend its landing gear, cooling panels, etc.

4.2. Latency

The steps described in **[4.1.]** are performed very fast. Thousands of queries are made each second as rules are applied, programs are loaded and unloaded, and data is sent between modules and the AIE.

Indeed, the faster the computer running the OSFS session, the better. Plus, DGIIIAI has multi threaded code, so that in in computers that have more than one singe CPU core operations become even faster.

Despite all this, the fact that KB operations are intensive and in order to keep things coordinated and avoid race conditions in relation to the application of rules, SQLC and external modules should

only exchange data with the AIE during step [4.1.2.], while *Status = 'getfromnetwork'*. This means that there is a certain latency, meaning that a fraction of time passes between the moment you send a command via SQLC and the processing of that command by the AIE.

You should not confuse this AI processing latency with the flight response of the ship. Factors like inertia, acceleration, etc. may also cause delays in the response of DGIIIAI. Like in the case of any aircraft.

It is no coincidence that one thing that you learn when you are after a flight license in the real world is that you should think always in anticipation because aircraft almost never react immediately to the pilot's inputs. Take for example, the latency or delay between the moment in which a pilot throttles up the engines of a plane, and the actual acceleration of the aircraft as a result.

So, pay attention to the latency of the AI system in addition to the time it takes for the ship to react. It is not a lot of time, plus the Delta Glider is a pretty nimble ship, but precise and safe flight requires taking into account what happens in every second in the air or in orbit.

5. Use

5.1. Installation

SQLC installs in a similar way that other similar modules to ~/modules/plugin, but in addition to the file SQLC.dll and its corresponding docs, it also installs a database in ~/databases/SQLC. It also requires that ~/databases/DGIIIAI/DGIIIAI.db is installed.

5.2. Functions

5.2.1. [DB]

Database. Lets the user connect to a specific database. With the **[DB]** function the user can switch between DGIIIAI and SQLC databases.

5.2.2. [CON]

Connect. Execute the connection to the database whose name was entered with the **[DB]** function and window dialog.

5.2.3. [DIS]

Disconnect. Disconnects the MFD from the currently connected database.

5.2.4. [SQL]

SQL query. Allows the user to enter a full SQL query as a string using a dialog.

5.2.5. [MON]

Mode on. Sets a mode – say, for the sake of an example, *mode_crs* - specified by the user on, which equates to sending to the AIE the SQL query *UPDATE sde_facts SET Value = 1 WHERE Item = 'mode_crs'*

Once you press [MON], a dialog appears and you have to enter the mode name that you want to activate, just by entering its name without the first part comprised of the substring *'mode_'*. For example, instead of entering *'mode_crs'* you will have to input *'crs'*.

5.2.6. [MOF]

Mode off. This performs the same query as **[MON]** except that it sends *Value = 0* instead of *Value = 1*. Once you press **[MOF]**, a dialog appears and you have to enter the mode name that you want to deactivate, just by entering its name without the first part comprised of the substring *'mode_'*. For example, instead of entering *'mode_crs'* you will have to input 'crs'.

Notice that **[MOF]** and other similar activation or deactivation functions allow you to send a full SQL query just by pressing the corresponding button, writing and abbreviated version of the mode name (with no *'mode_'* part), saving time and trouble, but you can send the same query using **[SQL]** if you like.

5.2.7. [PON]

Program on. **[PON]** and its companion **[POF]** work in a similar fashion to **[MON]** and **[MOF]** but

instead of activating and deactivating modes, they load and unload programs from table *sde_prg_rules* to *sde_rules*.

Normally this process is performed by the AIE itself, but in some circumstances you may want to control that process yourself.

When you press **[MON]** a dialog appears and you have to enter only the numbers that correspond to the program number and version that you want to load, without the string *'prg'*. For example, if you want to load program *'prg43.2'* you only have to input *'43.2'*.

5.2.8. [POF]

Program off. Performs the opposite operation to that of **[PON]**.

5.2.9. [ITM]

Item. This function opens a dialog that lets you enter the *Item* string that identifies a fact you want to operate with. For example, if you want to set manually a specific target heading for your ship during atmospheric flight, you will press **[ITM]**, enter the string *'tgt_hdg'* and later use **[VAL]** (see below) to enter the actual heading you want.

Always remember that **[ITM]** and **[VAL]** work together. You need to enter values using both functions in order to alter a fact.

5.2.10. [VAL]

Value. After entering the name of a fact that you want to operate with directly using [ITM], you need to give that fact a Value. This is what [VAL] is for.

Once the dialog opens, you need to enter a number. Following with the example presented in **[5.2.9.]**, once you entered *'tgt_hdg'*, if you want your ship to turn to a 90 deg heading, you should press **[VAL]** and enter *'90'*.

5.2.11. [EXE]

Execute. This is used to actually send data to the AIE. Whenever yo use, for example, **[ITM]** and **[VAL]**, that fact – changing data is not sent away to the AIE right away but is kept in a buffer.

This gives you time to enter data correctly without doing things in a hurry and confirming your intentions by pressing **[EXE]**. This sends the data and effectively clears the buffer.

Bear in mind that whenever you press **[EXE]** you are sending something to the AIE. What that something is depends on what is in the buffer at the time. So don't press **[EXE]** unless you mean it.

Also take into account that there might be some latency experienced whenever data is sent between modules. This might happen for several reasons like the hardware resources of your system, the size of the KB, the kind of tasks that are being performed by the CPU at the time, etc.

5.3. Procedures

These are some procedures that should be performed using the functions described in **[5.2.]**. In all these cases it is important to consider that depending on the kind of computer system in which you are running your OSFS session, you might experience some latency whenever data is sent between the MFD and the AIE.

5.3.1. Connect to a database.

The AIE onboard DGIIIAI starts working the moment the ship is loaded into OSFS' session.

However, if you call SQLC, in order to interact with the KB you need to connect to the database. Thus, you should perform procedure **[5.3.1.]** when you load the MFD for the first time and after you want to reconnect to the KB if you have performed **[5.3.2.]**.

5.3.1.1. Press **[DB]** and wait for the dialog box to open.

5.3.1.2. On the dialog box write ‘DGIIIAI’ to connect to the DGIIIAI.db database or ‘SQLC’ to connect to SQLC database.

5.3.1.3. Press **[CON]** to connect.

After calling this MFD for the first time in your Orbiter session, you should connect to DGIIIAI.db with this procedure.

If you want to switch databases, first disconnect from the current database you are connected to following **[5.3.2.]** and then perform **[5.3.1.]**.

5.3.2. Disconnect

5.3.2.1. Press **[DIS]**.

5.3.2.2. Wait until all pending DB operations are finished. This might take a second or so.

Before leaving a database it is recommendable to disconnect or close it. Otherwise Sqlite3 might leave some connections open and when you finish your OSFS session things might crash or even data might get corrupted.

After disconnection, you will have to perform **[5.3.1.]** to connect to a database again.

5.3.3. Send a raw SQL query

5.3.3.1. Press **[SQL]** and wait for the dialog box to open.

5.3.3.2. Write a full SQL query on the dialog.

5.3.3.3. Press **[EXE]**.

5.3.4. Activate a mode using [MON].

There are various ways to activate a mode, but for convenience and efficiency, you have the **[MON]** function.

5.3.4.1. Press **[MON]**.

5.3.4.2. Once the dialog appears, enter the name of the mode without the *‘mode_’* substring. For example, if you want to activate *‘mode_wpt’* you would just input *‘wpt’*.

5.3.4.3. Press **[EXE]**.

5.3.5. Deactivate a mode using [MOF].

There are various ways to deactivate a mode, but for convenience and efficiency, you have the **[MOF]** function.

5.3.5.1. Press **[MOF]**.

5.3.5.2. Once the dialog appears, enter the name of the mode without the *'mode_'* substring. For example, if you want to deactivate *'mode_wpt'* you would just input *'wpt'*.

5.3.5.3. Press **[EXE]**.

5.3.6. Load an AI program to *sde_rules* using [PON].

There are various ways to load a program, but for convenience and efficiency, you have the **[PON]** function.

5.3.6.1. Press **[PON]**.

5.3.6.2. Once the dialog appears, enter the name of the program without the *'prg'* substring. For

example, if you want to load *‘prg43.2’* from *sde_prg_rules* to *sde_rules* you would just input *‘43.2’*.

5.3.6.3. Press **[EXE]**.

5.3.7. Unload an AI program from *sde_rules* using [POF].

There are various ways to load a program, but for convenience and efficiency, you have the **[POF]** function.

5.3.7.1. Press **[POF]**.

5.3.7.2. Once the dialog appears, enter the name of the program without the *‘prg’* substring. For example, if you want to unload *‘prg43.2’* from *sde_rules* you would just input *‘43.2’*. Note that the records of the program are not deleted from *sde_prg_rules* when you execute this procedure.

5.3.7.3. Press **[EXE]**.

5.3.8. Alter a fact record’s Value field in *sde_facts.*

This method allows to change the contents of the Value field of any existing record in sde_facts.

5.3.8.1. Press **[ITM]**.

5.3.8.2. In the dialog box, enter the name of the *Item* to change (the string contained in *sde_rules*.

Item). For example, if you want to set *mode_hato* on using this method, enter *‘mode_hato’*.

5.3.8.3. Press **[VAL]**.

5.3.8.4. In the dialog box, enter the *Value*. For example, to set *mode_hato* on and activate it using this method, enter *1*. In some cases you might want to enter this value as an integer and in other cases as a real number. It is indistinct if you enter *1* or *1.00* in the case of modes because they operate as real numbers that implicitly are Boolean.

5.3.8.5. Press **[EXE]**.

6. Sources

6.1. Edronkin, P. (2019). DGIIIAI - Intelligent Delta Glider. [online] DGIIIAI. Available at: https://peschoenberg.github.io/DGIIIAI/ [Accessed 26 Aug. 2019].

6.2. Sqlite.org. (2000). SQLite Home Page. [online] Available at: https://www.sqlite.org/index.html [Accessed 26 Aug. 2019].

6.3. Schweiger, M. (2000). Orbiter 2016 Space Flight Simulator. [online] Orbit.medphys.ucl.ac.uk. Available at: http://orbit.medphys.ucl.ac.uk/ [Accessed 26 Aug. 2019].

7. Alphabetical Index

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W