

**Official Documentation**

Version 0.1  
Distributed under GNU/GPL license.  
Created by Sylvain Cloutier in June 2014

# Simple Data Fix Language (SDLF) – Documentation

## Miscellaneous information

### The author

My name is Sylvain Cloutier. At the time of writing those lines, I'm a developer at CGI in Quebec City, Canada. I have my OCP certification and I’m about to pass the OCE exam. I am also a member of the Quebec City JUG.

At CGI, I work on a team that supports hundreds of applications written in tens of different languages for a client in the manufacturing sector. The team is pretty much self-managed and the responsibilities of every developer are quite big as anyone can play the role of developer, analyst, architect or even DBA. We also imperatively do peer-reviews on every deliverable in order to maximize the knowledge transfer among the team. We have the quality of the deliverables in our top priorities.

I now want to share my professional experience with the open source community, simply because I use open source applications in my day to day job.

### The problematic

I personally think that the main problem in any custom applications, legacy or not, is the data storage. When people first came with the idea of relational DBMS, it was supposed to be a revolution as opposed to the mainframes storage or all types of document-based database like IBM’s Lotus Notes. The main argument why that was supposed to be revolutionary is that it was strongly structured. We have objects in our code and tables in the database, the different relations between the objects could be transposed into the database, parent/child relations were enforced by constraints, etc.. So we started to have RDBMS like Oracle, SQL Server, Informix or MySQL.

Now, the problem is that we are in the future of those days and we have to support all those applications built back then! We all know how difficult it is to make a change to the database model, or even simply change the RDBMS underneath, say from Informix to Oracle. Most of the time, cost matters so we simply don’t do it and find an ugly workaround. From now on, the application is considered as legacy and we plan to decommission it. But it still does the job, so we still have to maintain them.

I’m not saying here that this is the only way we can get an application to become legacy. There are plenty of reasons, but let’s concentrate of this one for the purpose of this document. The solution the community proposed to resolve this it to use other kinds of DBMS. One on the most popular one is the document-oriented DBMS, like MongoDB or Cassandra, also known as Big Data, just like the old IBM’s Lotus Notes. What are their arguments? They kindly say that today’s machines are now cheap. Storage is cheap, RAM is cheap, fast processors are cheap, so why not abuse them? I read somewhere: Software engineers work all day to destroy the work of the hardware engineers. Where has the will to produce software with the smallest footprint possible gone?

That being said, I hear you asking: Is SDFL resolving that those issues? Not at all! We must admit it; we are either stuck with the technology choices made by the people who originally designed the applications, or we are willing to pay or charge huge amount of money.

### Purpose of the language

SDFL is a tool, a layer above the DBMS. In Java, we have JPA between our code and the database, in .NET, it’s called Entity Framework. What is the main purpose of those frameworks, apart from offering data caching and other things that simplifies data access? It offers a detachment from the proprietary SQL.

Outside any programming language, I couldn’t find anything that was providing a total abstraction of the DBMS and which would allow us to perform queries, data extracts, data load, stored procedures, DDL statements, etc.. Of course, there are data modeling applications that allows to design the DB and then generate the SQL DDL code, but I find them somewhat limited when we are in the maintenance mode. Moreover, I kindly challenge you to show me an MRD of an application build 20 years ago that is still representative of the actual database.

That is the purpose of SDFL. I provide an SQL-Like language, hopefully easier to learn and read, which can be compiled in proprietary SQL or executed by the SDFL Executor that provides a larger set of commands.

*\* The source code of all components of SDFL is available on SourceForge. \*  
\* SDFL is released under the GNU/GPL. \**

## Language specific information

### File extensions

SDFL source files can have different extensions. The “.sdfl” and “.sdf” are basic SDFL source code files. Both will be compiled the same way. On the other hand, “.sdfp” files will be treated as SDFL+ sources files. Those sources will be compiled into a special format destined to be run by the SDFL Executor.

### General syntax

The Simple Data Fix Language was originally designed to be as clear as possible. I have tried to make the different statements as easy to read as possible. I also tried to keep the syntax and grammar of other popular languages such as C++ and Java in order to reduce SDFL’s learning curve and I introduced new keywords and declaration structures so it becomes less verbose.

In SDFL, the different statements must be separated by semi-colon. All whitespace characters are ignored except, obviously, between keywords.

### Reserved keywords

Here are listed all keywords used by the SDFL compiler. Those keywords cannot be used as variable names or as any identifier. On the other hand, SDFL is case sensitive, so “Package” is not a keyword, but “package” is. Though the compiler allows the use of such identifiers, it is strongly recommended using names that differ from the keywords for better understandability.

|  |  |  |  |
| --- | --- | --- | --- |
| **in** | **package** | **create** | **datafix** |
| **insert** | **delete** | **import** | **into** |
| **using** | **template** | **with** | **headers** |
| **update** | **if** | **not** | **exist** |
| **and** | **or** | **only** |  |

### Comments

SDFL comments are based on C/C++ comments, meaning that both C++’s single and multiline comment delimiters are available. For single line comments, use “//”. Everything after this delimiter until the end of the line will be ignored by the compiler. For example:

// This line is a single line comment  
**in package** MY\_DATA\_FIX;

Multiline comments start with “/\*” and end with “\*/”, like:

/\*\*  
 \* This is a multiline comment.  
 \* Created by Fragmatyc  
 \*/  
**create datafix** INIT – Initial load of the data

## SDFL Statements

### Organization statements

SDFL offers a set of statements that are only destined to organize the code properly. SDFL source files are compiled into deployable package that are defined by those set of statements.

#### “in package” statement

The **in package** statement is a mandatory statement that must come at the first line of code in any deployable packages, apart from any comments. The purpose of this statement is to tell the compiler in which package the SDFL code is. Usually, the compiler creates a folder with the package identifier and stores the compiled code in it.

The only parameter this statement accepts is the package identifier and is mandatory. It cannot be a String and must not contain spaces of special characters. Only alpha-numeric characters and underscores “\_” are accepted.

Example:

// This creates a package called “PIZZAPP\_FIX”  
**in package** PIZZAPP\_FIX;

#### “create datafix” statement

This statement is used to create a module in the data manipulation package and simply allows separation between logically related scripts. The compiler usually creates a separated folder to store the compiled files. There must be at least one data fix per package.

The **create datafix** statement takes 2 parameters. The first one is the identifier of the data fix. Like the package name, only alpha-numeric characters and underscores “\_” are accepted. Then, there is the description. The identifier and the description must be separated by a dash “-“.

Example:

// This creates a new data fix called INIT  
**create datafix** INIT\_DDL – Create the tables;

// This one doesn’t have a description  
**create datafix** INIT\_LOAD;

### Data modifications statements

#### “import” statement

One of the things SQL doesn’t provide is a way to simply load data from a CSV or XLSX file. SDFL offers the import statement to achieve this.