*SOL Software d.o.o.*
*www.sol.rs*

**Public**

# SOLoist Automatic Execution of SQL and Java Alter Scripts

| |
|---|
| **Project:** SOLoist V4 |
| **Document Type:** Project Documentation (PD) |
| **Document Version: 1.0** |
| **Date:** 01.06.2014 |

# Introduction

This document describes how SOLoist V4 (starting from version 1.0.7) can help in execution of SQL (and Java) alter scripts for database schema evolution. This is only about execution of alter scripts and not about their creation. Creation of SQL alter scripts is a matter of *Database Evolution* tool explained in a separate document. Creation of Java alter scripts (scripts that create or modify SOLoist object space) is not in the scope of this document.

# The Idea

The main objective behind this extension is to help SOLoist developers store, keep track, and execute SQL (and Java) alter scripts for database schema evolution (due to the evolution of UML model) in a controlled and easy-to-use way. After a new version of SOLoist application is installed on the target environment, SOLoist performs a series of tasks in order to adjust the old database schema (and the data) to the requirements of the new version of the application. Every new version of application may have one SQL alter script and one Java alter script. SQL alter scripts are meant for schema alterations. Sometimes, in simple cases, they can be used for data alterations as well. On the other hand, Java alter scripts are meant for complex data alterations, where it is much easier to create data alter script in Java than in SQL. Simply stated, after the application is installed on the target environment, SOLoist will first check the current database version and then perform all subsequent SQL alter scripts and Java alter scripts in order to adjust the database to the newly installed application. Everything is logged in a special table that keeps track of every database alteration.

# The *SchemaChange* Table

SOLoist keeps track of every database alteration and records it in a special table named *SchemaChange*. Every row in this table is a record of a single database alteration. The table has the following columns:

| Column | Description |
|---|---|
| version | The version of the database schema. |
| applied | When the alteration script has been applied? |
| description | The description (the comment) of the alteration script. |
| alterscript | The SQL alter script itself (as a blob). |
| alterhook | The Java alter script itself (as blob). |

This table is not only for logging the performed database alterations, but also for checking the current version of the database schema. The current version of the database schema can be obtained with:

```
SELECT MAX(version) FROM SchemaChange
```

This is how SOLoist checks the current version of the database schema. Based on this, SOLoist will execute only those alter scripts that were created after this version of the database schema. For example, if SOLoist determines that the current version of the existing database schema is 17, it will try to execute only those alter scripts that came after, which means only those alter scripts that are versioned as 18, 19, 20, etc. This mechanism allows developers to install new version of an application with ease, no matter what the difference between the existing application and the new application version is. SOLoist will do everything automatically and adjust these two to fit.

## Naming and Other Conventions

There are some requirements (or conventions) that must be met in order to allow SOLoist to perform automatic execution of alter scripts correctly.

The SQL alter scripts should all be kept in the same folder within a project, one script for every version of SOLoist application. Every alter script should be named:

**`alter-X.sql`**

Here, X is the number (integer) of the database schema version. Every subsequent SQL alter script should have the X number higher than all previous SQL scripts. The easiest way would be to have SQL alter scripts named like this: alter-1.sql, alter-2.sql, alter-3.sql, and so on. Here, the alter script *alter-3.sql* is newer than alter script *alter-2.sql*, and so on.

Similarly to SQL alter scripts, Java alter scripts should all be kept in the same Java package. They should be named like this:

**`AlterXHook.java`**

Again here, X is the number (integer) of the database schema version. Every subsequent Java alter script should have the X number higher than all previous scripts. The easiest way would be to have Java alter scripts named like this: Alter1Hook.java, Alter2Hook.java, Alter3Hook.java, and so on. Here, the alter script A*lter3Hook.java* is newer than alter script A*lter2Hook.java*, and so on. Every Java alter script should extend the *ADbInitHook* abstract class and implement its *handle* method.

A new version of a SOLoist application should have zero or one new SQL alter scripts and zero or one new Java alter scripts. For example, in case of both database schema modifications and object space modifications, the new version of a SOLoist application could introduce two alter scripts: alter-27.sql (for database schema modifications and simple SQL-based data modifications) and Alter27Hook.java (for complex Java-based data modifications).

This extension allows for attaching comments to every new version of the database schema. The comment should explain what was modified in the database schema or in the data. In order to set a comment for a new version of the database schema, the comments.properties file should be put alongside alter-X.sql files. The format of this file is shown below:

```
#Comments regarding alter scripts
alter-1=Initial database script.
alter-2=Added attribute X in class Person. All objects set to have value “something” for this attribute.
alter-3=Added association between classes Person and Company.
```

Every time new alter-X.sql file is added into the project, a corresponding comment should be added as well. This comment will be set in the “description” column of the SchemaChange table for the corresponding database version.

## Configuration

This SOLoist extension is configured in the web.xml file. To switch it on, this should be added in the web.xml file:

```
<listener>
```

```
        <listener-
class>rs.sol.soloist.dbevolution.dbinitialization.DBInitContextListener</listener-class>
</listener>
```

If this extension is turned on, SOLoist will search for SQL alter scripts in /WEB-INF/classes folder by default. This means you should make sure these files end up there after you create your web application war file. You can also configure this path by setting the context parameter like this:

```
<context-param>
        <param-name> DBINIT_DB_DIR</param-name>
        <param-value>/WEB-INF/classes/mysql</param-value>
</context-param>
```

In order to use Java alter scripts, Java package FQ name must be added in web.xml file like this:

```
<context-param>
        <param-name>DBINIT_HOOKS_PACKAGE</param-name>
        <param-value>rs.sol.myproject.javahooks</param-value>
</context-param>
```

To perform the database alterations, SOLoist needs database access parameters (url, username, password, etc.). By default, SOLoist will use the db.properties file from the project for this. However, since this extension is required to execute major database alterations (like creating tables, indexes, etc.), separate access parameters could be specified for it. In order to set separate database access parameters for this extension, this should be added in web.xml:

```
<context-param>
        <param-name> DBINIT_DB_PROPERTIES</param-name>
        <param-value>db-init.properties</param-value>
</context-param>
```

If a SOLoist application is to be installed on a target environment for the first time with this extension, there is an additional parameter that may be specified if necessary. In this case the problem is in the fact that SchemaChange table does not exist in the existing database, hence SOLoist cannot determine what alter scripts to execute because it cannot determine the version of the existing database. In this case you may want to tell SOLoist to which version the existing database schema corresponds to. This would be taken into account when SOLoist decides what alter scripts to execute. So, if you tell SOLoist that the existing database corresponds to version 15, SOLoist will execute only those alter scripts that are newer than version 15. If you do help SOLoist with this, it will execute all found alter scripts (as if it was told the existing database version is zero). In any case, if SOLoist does not find the *SchemaChange* table (and this extension is turned on, see bellow how to turn it on), SOLoist will create this table automatically. It is very important to note that all this is relevant only in case you install SOLoist application for the first time with this extension turned on. To tell SOLoist the current version of the database use this context parameter:

```
<context-param>
        <param-name>DBINIT_VERSION</param-name>
        <param-value>15</param-value>
</context-param>
```

Finally, there is one last configuration parameter. Sometimes when writing stored procedures or triggers in SQL alter scripts, this extension finds difficult to determine where the stored procedure ends or where the statements within stored procedure end. This is why separate (different) delimiter character should be used to end the store procedure itself as opposed to

statements inside the stored procedure. The delimiter can be set for every alter script individually. This is done like this (for alter script 2 for example):

```
<context-param>
	<param-name>DBINIT_ALTER_DELIMETER_2</param-name>
	<param-value>//</param-value>
</context-param>
```

## Database Support

Currently, this extension supports Oracle, SQL Server, and MySQL databases.