# usp\_data\_correction (Data Correction Stored Procedure)

## Purpose

This stored procedure tries to encapsulate a number of techniques used when doing data corrections.

Backing up the table, verifying changes against the backup table, the transaction, auditing, replication, etc. With less code to write and simpler flowcharts, the hope is that it will be used to save time. It will

also guarantee consistency; what to put in the audit trail, whether to backup inside the transaction, the use of Goto statements, etc. There are certain rules that need to be followed to use it and the input validation tries to ensure those rules. The second hope is that it will grow to accommodate different cases of data corrections. The examples provided attempt to illustrate some of the different cases of data corrections that it can accommodate.

## Data Correction Cases/Techniques

**UPDATE ONLY**

In this case @replication\_ind = ‘N’ and @insert\_set\_sql should be non-empty.

**REPLICATION ONLY**

In this case, the user should make all records have @replicate\_rec\_ind = ‘O’. The stored procedure will then ignore @insert\_set\_sql

**UPDATE THEN REPLICATE**

In this case, @replication\_ind = ‘Y’ plus the user should make all records have @replicate\_rec\_ind = ‘Y’ and @insert\_set\_sql should be non-empty.

**UPDATE THEN REPLICATE LATER**

As above, except a persistent table is used as the control table instead of a global temporary table, and replication is performed in batches. The second time the script is run, @replication\_ind = ‘P’ with a @rep\_batch\_limit

**TARGET REPLICATION**

If replicating, then each record must have a target code.

-If replicating to all locations, the target should be NULL or an empty string.

-For location specific replication, the target code should be the store/location code.

-For retail chain specific replication, the target code should be ('^' + retailchain\_code)

Note: The procedure orders by rep\_target, so that as many dcmtaskcontents are packed into each task as possible (up to 198)

**MULTIPLE TABLES**

The user may also want to update multiple tables at once in a transaction and/or may want to replicate from many tables. This means insert more records into the control table with different table names.  
  
**BACKING UP TABLES**In addition to the @control\_backup\_table, all distinct table\_name/backup\_table pairs will be backed up.

**AUTO UPDATE/DELETE/INSERT**

The user may need to create temporary tables before the update. This must be done dynamically, by putting the declaration and population of the temp table inside the select sql parameter.

**UPDATE PER ROW**

The user may need to execute statements in a loop, rather than just with one statement. In this case, the looping must be done dynamically inside the transaction sql parameter.

**AUTOMATIC COLUMN MATRICES**

The number of columns in a table that will be replicated can be calculated internally, rather than being in input. This will make every data correction version independent.

## Testing

Each example should create the backup table, makes the correct transaction and insert into the dcmtask tables with the correct status, column matrices and business objects, etc.

## File Structure, Review and History [\\rdsvr\Public\Support\SQL Server Scripts\usp\_data\_correction\](file:///\\rdsvr\Public\Support\SQL%20Server%20Scripts\usp_data_correction\) should contain a history folder and a review folder. If changes are made the new script goes in the review folder. After review the old script goes to History and the reviewed script becomes the new one.

## Thoughts And Unanswered Questions

**Concurrency**

I'm not entirely sure whether there are any issues with concurrency. One thing to note, is that the user shouldn't always use the same name for the @control\_table. Otherwise, there could be a problem if 2 RD employees accidently do a data correction at the same time, using the same @control\_table global temp table, e.g. ##my\_call.

**An Error Message**

If the transaction is not committed inside the stored procedure, an error message appears that says this shouldn't be done inside a stored procedure. This is something we need to be able to do. Nonetheless, it seems the transaction can still be committed or rolled back after executing the procedure. Is this an issue? Is there another way? Can this warning be suppressed?

**SQL Injection and security**

How do we modify the security on the stored procedure so that only our database user can modify it/use it? Even though the user cannot use the stored procedure through a client application at this time, since it contains dynamic SQL, it's worth checking for SQL injection. However, at this time the code is commented out as I haven't got it to work.