

## A Plug and Play Data Entry Audit Trail System

Tyler Witko, Pfizer Inc., New York, NY

### Abstract

A pharmaceutical clinical data entry system requires many functions to be both useful and compliant with regulatory guidelines. One important function is a robust audit trail mechanism which documents changes to regulatory data throughout a database lifetime. This audit trail should capture the who, what, when, and why of changes made to data. It should provide useful reports and the ability to 'roll back' the database to any previous state.

This paper describes a 'generic' audit trail system written in SAS<sup>®</sup> Screen Control Language, (SCL), and implemented through the SAS/FSEDIT<sup>®</sup> product that provides the audit functions listed above and can be used on any SAS System data set with absolutely no programming. This system captures all the required information and provides interactive rollback/forward options to step through a record's editing history. As well, any previous state of a record can be committed to the data set at any time.

This system is 'plug and play' in that the SCL code can be included into any customized SAS/FSEDIT screen definition and used as is with no modifications.

### Introduction

The nature of clinical trials does not facilitate the deployment of a standard database structure across studies for data input and editing. While key study and patient information can be stored in a common format, study specific data require customized data structures. This is due to the fact that data items relevant to one study may have no significance in another.

While different clinical studies may propagate heterogeneous database structures, a standard audit trail mechanism and interface can be deployed on any SAS data set. Several benefits of implementing a standard audit trail system include:

- Provide a single mechanism for tracking editing changes.
- Standard program(s) used to roll back a database to a previous state.

- Common user interface to the audit trail makes its use less intrusive.
- Improved confidence in database integrity.
- Disaster recovery option providing the audit trail remains intact.

Other benefits of an administrative nature include:

- Historical evaluation of editing trends, (are specific entry errors more common and can they be reduced; does one type of editing screen layout promote more accurate/efficient data entry).
- Performance monitoring of data entry staff.

While the SAS/FSP<sup>®</sup> products with SCL provide a powerful data editing environment, implementing an audit trail usually requires customized programming for each data set. Often the resources required for this customization are not available.

### Functions

The need for audited data entry and lack of programming resources motivated the design and development of a generic SCL program that would operate on any SAS data set. The single constraint on data set structure is detailed in Design below.

This general goal gave rise to the following functional specifications:

- Document any changes to existing data, (what, when, who, why).
- Produce reports of the documented changes.
- Be able to roll back a record to any previous state.
- Be as unobtrusive to the user's environment as possible, (transparent).
- Implement system on any SAS data set by a non-technical user.

### Design

The essential components of a usable audit trail provide a means to identify what data has changed, the original, (pre-change), value, and when the change occurred. Of these three items, identifying the record changed was the most challenging. Two approaches were examined.

The first approach is to identify the key fields for each data set and capture the fields' values for each record audited. This approach required an additional data store to hold the identifying information and presented complications should one of the key field values need to be changed. Preventing audit trail orphans could be complex and time consuming.

The second approach, (and the one selected), is to have one common unique key field for each data set whose value is determined by the system and cannot be changed. This record number field, (recnum), is required in all database tables using the audit trail software and is one of the primary keys between the edited table and the audit trail table.

Use of an immutable record number field greatly simplified parent-child relationships and allows editing changes to any other database item without risking audit orphans. The recnum field is assigned a value for any record added to the table.

### Logical Design

The logical design of the audit system is straightforward: Each database record can have many related records in the audit trail table. Each audit trail record has only one parent record in the database and it maps back to one column in that parent record.

There is one wrinkle that was accommodated using an 'action' field in the audit trail to identify deleted records. The special case is due to the record number generating mechanism. If a record is added or duplicated the SCL varstat function is used to determine the current maximum value of recnum and assigns the next value to the new record.

The difficulty arises if records are written to the audit trail and then the database record is deleted. When another record is added, the deleted recnum value will be reused and may cause confusion with existing audit trail records. This confusion is avoided by inserting a 'Delete' record in the audit trail to indicate the end of a specific recnum value's life. The 'Delete' record acts as a barrier to mistaking a deleted record's audit trail as belonging to an existing record.

### Physical Design

The physical design of the audit system was also rather straightforward. The audit trail table has fields to hold database record identifying information and data change related information.

All database data values are stored in the audit table as character data types. That is, numeric data values in the edited table are converted to their character equivalent for storage in the audit table. Numeric items with a date or time format are stored as the formatted character literal, (eg. '10aug1995', '10:00:00'). While this decision required storing the item data type and format, (for conversion from character back to numeric), it improved the usefulness of the audit trail reports.

A second design decision was to store both the original and new, (changed), data value in each audit record. While this created larger record size and redundant data, (a record's new value is either in the database record itself or in the next audit trail record), it made for reports that were easier to read and simplified the rollback/forward options. See figure Table Audit for physical structure.

**Table Audit**

<b>Name</b>	<b>Type</b>	<b>Length</b>	<b>Label</b>
dsname	C	8	Data set Name
recnum	N	8	Parent Record No.
varname	C	8	Variable Name
vartype	C	1	Variable Type
varfmt	C	16	Variable Format
oldvalue	C	100	Original Value
newvalue	C	100	New Value
reason	C	40	Reason for Change
action	C	1	Action
lastmod	N	8	Date of Change
userid	C	15	User id

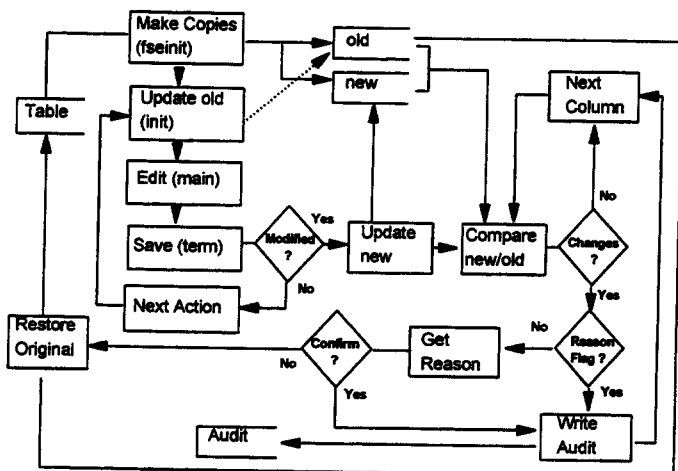
### Audit Trail Logic

The audit trail subsystem is invoked whenever an fsedit session is started. At a high level the audit system functions in the following manner:

- Create work tables to hold old and new values
- Update OLD table
- Edit database table
- Update NEW table
- Compare OLD and NEW tables
- If changes detected append audit trail

A more detailed description follows, (see Audit Trail Data Flow for diagram):

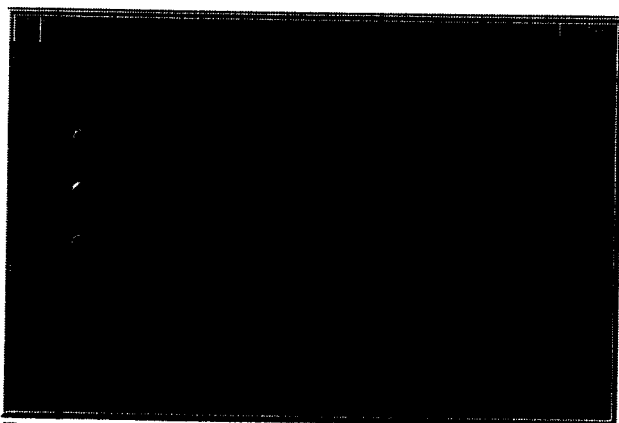
Before the fsedit screen is displayed, (fseinit), two single record copies of the edited table are created with all values set to missing. The tables are named WORK.OLD and WORK.NEW. Before each record is displayed for editing, (init), its data values are written to OLD.



**Audit Trail Data Flow**

When the user invokes a save action, (File-Save, Page Down, Add, etc.), and modification(s) have been detected the following events occur, (term).

The values currently displayed are written to the NEW table. The value in each field of OLD is compared to its corresponding value in NEW. If the values are different and the original value is not missing and the 'reason for change' flag is not true, the reason program is invoked to capture the reason for editing change, (see Reason for Change).



**Reason for Change**

When a reason is selected and confirmed, the reason flag is set to 1, the current column values are written to the audit table and the next column pair, (from OLD and NEW), are compared. The reason flag acts to capture the reason for change only once for each record edited. By removing the flag check, a reason for each field can be captured.

It is the combination of using single record copies of the edited table and updating the copies with original and new values of the currently edited record that provides table structure independence. There is no need to know any column names or the

current observation number to compare original and new column values. The recnum value identifies the record and column names are determined with the SCL varname function.

The only required audit trail data not available in the fsedit session are the name of the table being edited and the user id. These values are captured from symbolics set prior to the fsedit command.

**Cancel Changes**

During display of the Reason for Change panel the user may decide to Cancel the editing changes. Selection of the Cancel button issues a warning message then the original data values, (stored in OLD), are retrieved.

**Duplicate Records**

The audit trail is also invoked immediately after a duplicate command. This event is captured by the obsinfo('new') function and checking the last command issued, 'DUP'. In this event, the audit trail unconditionally writes all the current non-missing column values to the audit trail.

The reason for this is to document this record's values at this point in time. This is critical if the database needed to be restored to the state immediately following the DUP command.

**Roll Back/Forward**

Since all changes to a record's values are stored, they are available for recall. The on screen undo and redo options recall records from the audit trail in chronological sequence and assign screen fields the values. Here is where storing the column's data type and format are required, (to convert stored character values to numeric screen values).

There was a significant technical hurdle with the roll back mechanism. Because the SCL has no knowledge of the edited table's column names, assignment statements could not be used, (eg. *pat = getvar(ds\_new, varnum(ds\_new, 'pat'))*). What we needed was a place to put the values recalled from the audit trail and then fetch that data and update to the current edited record using the automatic links provided by *call set(ds\_id)*. The table WORK.NEW was chosen to hold the audit trail values.

Fetching WORK.NEW's data vector was not a problem. However, because a *call set* link to the database table was previously made, any attempts to update WORK.NEW with audit trail data resulted

in writing the values currently displayed on the FSEDIT screen.

We needed to break the automatic link between the edited table and WORK.NEW. This is done by closing WORK.NEW and then re-opening it. By default, an opened table is not linked to the edited table. Now we could fetch the data vector of WORK.NEW, assign audit trail values with *putvarc* or *putvarn*, and update NEW with *update*. We are not done yet. We still need to get the values from NEW to the fsedit screen. This is done with *call set* and *fetch*.

Programmatically the process, (simplified), looks like:

```
* subset audit table to current recnum, then...;

if attrn(ds_aud, 'any') then do;
  rc = close(ds_new);
  ds_new = open('work.new', 'U');
  rc = fetch(ds_new);

  do while(^fetch(ds_aud));
    rc = putvarc(ds_new,
      varnum(ds_new, varname), oldvalue);
  end; * do while(^fetch(ds_aud));

  rc = update(ds_new);
  call set(ds_new);
  rc = fetch(ds_new);
end; * if attrn(ds_aud, 'any');
```

The undo and redo options are useful for verifying changes were made to a specific record or recovering from accidental changes.

### Reports

Audit trail related reports currently provide listings of a current record's history sorted by a choice of date of change, userid, or reason for change.

### SCL Implementation

To provide a simple method of installing the SCL source code in each fsedit screen definition it is stored as an external file. A customized fsedit screen is opened for each table and the external file is included into the SCL program window and compiled. These actions have been defined in a function key and can be processed with one keystroke. The key definition is:

```
mod; 3; clear; inc 'path\fsedit.scl'; end; end;
```

Our actual implementation is a bit more complex as this is one module in a larger SAS database management system.

### Conclusions

Our experiences to date with this audit trail system while limited, have been very positive. It provides a superior level of control over manually edited databases with little impact on response time. For a table with seventy columns, the compare/write to audit actions are barely noticeable, (under one second).

Data entry staff find the Reason for Change panel very painless and appreciate the ability to verify on line that a specific record has been changed as required.

Data managers appreciate the audit reports which facilitate the verification of database updates without having to print all data and sift through unchanged records.

This audit trail mechanism demonstrates the power and flexibility of the SAS System to deliver mission critical applications that retain the ability to tailor functions to an individual organization's needs.

### Author Contact

Pfizer Inc.  
Tyler Witko  
235 E 42 Street MS 219/6/11A  
New York, NY, 10017  
(212) 573-1776

E-Mail: 74004.1076@compuserve.com

### Trademarks

SAS, SAS/FSP, SAS/FSEDIT are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries.  
® indicates USA registration.