RAD with SAS® Software:
Strategies, Techniques, and Tips
Samuel Berestizhevsky, Applied Statistics & Systems Consultants, Israel
Tanya Kolosova, Applied Statistics & Systems Consultants, Israel

ABSTRACT
How can an application meet future needs when the business situation constantly changes? How fast can new conditions be met? The answers to these questions depend on application development technology [1]. In this paper we describe strategies and techniques intended to reduce applications development time and simplify modifications [2]. We show how to:
1. use relational technology to define an application data model and generate application objects;
2. define an application operations model specifying data manipulation and data analysis processes, and create reusable “user-written” macros;
3. define data authorization access to application objects.

INTRODUCTION
Application user expectations for new applications include short delivery lead-times, ease of use, and increased flexibility. In response to these challenges, application designers are adopting rapid application development techniques. In this paper we describe one of these techniques that lets the application designer specify applications visually, in terms of their functionality, and permits the development of mission-critical applications without programming.

The main principles of this approach are:
• Application design is perceived as data and nothing but data. This means that the application design is defined in a set of specially structured tables and is stored, updated, and managed in the same way as ordinary data.
• Application activities are stated in terms of what must be done, but not how to do it. An application activity can be imagined as a definition of states, and messages allowing transition from one state to another. Of course, the states and messages are stored, updated, and managed as ordinary data.
• The application is managed from a single control point.

The table-driven environment is the most fundamental aspect of the described approach. The heart of this environment is the set of specially structured tables forming the data dictionary. The data dictionary contains a variety of information concerning application objects and operations, such as data structures, application activities, and so on.

The proposed approach to applications development and maintenance can be used by the designer, SAS programmer and user of SAS applications in the following way:
1. the designer and programmer develop and produce the table-driven environment for generating various applications; the designer is involved into design of such an environment and the programmer is involved into the writing of programs that support this environment.
2. the designer and user design and develop their applications in the created table-driven environment without code writing; as a result the user gets really working applications generated by the table-driven environment.

HOW TO DEFINE AN APPLICATION DATA MODEL
Application design involves writing definitions of the application objects in the tables of the data dictionary that form the table-driven environment [2]. We must fill in the tables of the data dictionary with the following information:
• locations of application parts
• application tables
• columns of application tables
• relationships between application tables
• messages.
The Library, Object, Location, Property, Message, Link, Format, Comoper, Commacc tables form the data dictionary for application data model definition (see Figure 1).
In brief these tables are intended for:
• the Library table lists SAS library specifications for an application
• the Object table lists application tables and the names of corresponding SAS data sets where their data are stored. The Object table also defines application table titles and screen form types for visualizing and updating the data stored in SAS data sets
• the Location table lists application tables and SAS libraries, where SAS data sets corresponding to the tables are kept
• the Message table lists application messages
• the Property table specifies the properties of application tables. It lists the columns of each
application table and defines the following information for each column:

- name of the table of which that column is a part
- column name
- column data type
- column length
- column domain
- column location on the screen form, etc.

- the Format table is intended to specify user-defined picture formats.
- the Link table specifies pairs of application tables linked by foreign keys
- the Composer table defines communication operations between SAS libraries defined in the Library table
- the Commacc table defines the communication access methods for different parts of an application.

Because all the tables of the data dictionary work together and are affected by changes in associated tables, we should carry out our work in a certain order. The data model definitions for an application can be defined in six steps. These steps are shown on Figure 1.

HOW TO GENERATE APPLICATION OBJECTS

While the designer works with the contents of the data dictionary tables, the programmer works with the structures of those tables. While the contents of the data dictionary tables can be continuously changed (in accordance with the changes of an application requirements), their structures remain unchangeable. The data dictionary tables constitute the table-driven environment, and once developed programs processing these tables generate the wide variety of applications. The process of generating application objects (SAS data sets, formats catalogs, data entry screens) from the meta data of the data dictionary tables is called a bootstrap process. Such a process is shown on Figure 2.

HOW TO DEFINE AN APPLICATION OPERATIONS MODEL

Data manipulation and data analysis processes form the dynamic part of an application. This part represents application operations that are performed on the application data. Application operations may be considered in terms of events. If an event occurs or is about to occur, data manipulation or analysis operations are executed. Each application is defined in terms of an object and an operation. In other words, an application encapsulates both the data (objects) and the valid procedures (operations) that can be performed on the data. An operation that can be attached to a specific object is called object-dependent. Operations are optional; application objects need not have operations associated with them.
Object-dependent operations

An object-dependent operation cannot be directly called or executed; it is automatically executed by the application as a result of an action - data modification to the associated application object. It means that once an object-dependent operation is created, it is always executed when its “firing” event occurs (that is, updating or saving tables of an application object). Two options exist for when an object-dependent operation can execute: before or after the “firing” event occurs. For example, an update of a table is the event that invokes a table’s attached operation and it executes after the “update” occurs. In opposite, a save of a table is the event that invokes a table’s attached operation and it executes before the “save” occurs. An operation may itself perform update of another application table. Therefore, an operation is “fired” by data modification but can also cause another data modification, thereby “firing” yet another operation. When an operation contains update logic, the operation is said to be a nested operation. The ability to nest operations provides an efficient method for implementing, for example, automatic data integrity.

By implementing object-dependent operations, scheduling problems can be eliminated, because the operation is “fired” when the corresponding event occurs. We need not remember to schedule an activity to perform the logic in the operation. It happens automatically by virtue of being in the object-dependent operation.

Object-independent operations

An operation that is not attached to a specific object is called object-independent. Object-independent operation is not event-driven and is executed according to a user’s requests. An object-independent operation may contain within itself update of an application table. Thus, such an operation can “fire” the object-dependent operation attached to the updated table. The set of object-independent operations forms the user operations model.

Both object-dependent and object-independent operations can be stored in the tables of the data dictionary as easily as the data is stored there.

Application development process

The application operations model (consisting of object-dependent and object-independent operations) facilitates the application development process in a number of significant ways:

- it may not be necessary to develop an application program (in the traditional sense of the term) at all
- the application can be developed quickly and easily, without programming in the conventional sense
- when it is necessary to write a conventional program, it is easier to write, requires less maintenance, and is easier to change when it does require maintenance,

than it would be in other application development approaches
- the application development cycle can involve a great deal more prototyping than it used to:
  - a first version can be built and shown to the intended users, who can suggest improvements for incorporation into the next version
  - as a result, the final application does exactly what its users require of it
- the overall development process is far less rigid than it used to be, and the application users can be far more involved in that process.

Data dictionary for data manipulation and data analysis definition

Designing an application operations model, like designing an application data model, is iterative by nature. The designer, who deals with application objects and operations, must not define all the operations for all the objects at the same time. Table-driven strategy enables the designer to reach the final definition of the application operations model gradually.

The operations model for an application can be defined in six steps. These steps are shown on Figure 3.

Defining manipulations

The PMan table (see “Step6. Defining manipulations” on Figure 3) defines manipulations that must be performed on the input operation tables. Manipulations defined in the PMan table describe data manipulation and analysis actions. To describe an action we use SAS statements (like DO-END, IF-THEN, OUTPUT, etc.) and “user-written” macros. “User-written” macros are created by the programmer, based on the needs of the designer, and may use SAS functions, SAS macro functions, and SAS procedures. Such “user-written” macros give unlimited flexibility in using the SAS System resources to perform data manipulation and data analysis. For example, definition of the manipulation identified as “sameval” is shown in the PMan table like this:

<table>
<thead>
<tr>
<th>MANID</th>
<th>MANORD</th>
<th>EXPRESSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>sameval</td>
<td>1</td>
<td>if %same VAR1 then do;</td>
</tr>
<tr>
<td>sameval</td>
<td>2</td>
<td>VAR2 = VAR3;</td>
</tr>
<tr>
<td>sameval</td>
<td>3</td>
<td>VAR3 = %previous VAR3;</td>
</tr>
<tr>
<td>sameval</td>
<td>4</td>
<td>output;</td>
</tr>
<tr>
<td>sameval</td>
<td>5</td>
<td>end;</td>
</tr>
</tbody>
</table>
HOW TO GENERATE DATA MANIPULATION AND ANALYSIS PROCESSES

Meta data defining the “sameval” manipulation in the PMan table of the data dictionary includes so-called “user-written” macros such as %same, and %previous. The following programs present examples of “user-written” macros, developed by the programmer according to the designer’s requirements.

```
/*
PROGRAM Previous.
DESCRIPTION Stores the previous value of the column.
USAGE \%previous(varname);
PARAMETERS varname - is the name of the table column.
*/
\%macro previous(varname);
\%let prevar = %substr(&prevar, 1, 8);
\&prevar
\%mend previous;
*/
```

The data dictionary macro program performing manipulations specified with “user-written” macro uses source code of such a macro stored in the external file, on the one hand, and meta data from the specially structured PMacro table (described below), on the other hand, to resolve this macro. Generally, the processing of “user-written” macros is demonstrated on Figure 4.

Environment for “user-written” macros executing
The PMacro table (see Figure 4) specifies the names of the "user-written" macros, their properties and environment for "user-written" macros executing. The columns of the PMacro table are defined as follows:

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MACRO</td>
<td>Character</td>
<td>9</td>
<td>Name of the &quot;user-written&quot; macro</td>
</tr>
<tr>
<td>PROPERTY</td>
<td>Character</td>
<td>1</td>
<td>Macro's properties: O - executes out of DATA step, I - executes inside of DATA step.</td>
</tr>
<tr>
<td>PREMACRO</td>
<td>Character</td>
<td>9</td>
<td>Name of the macro that must be executed before the &quot;user-written&quot; macro.</td>
</tr>
<tr>
<td>PSTMACRO</td>
<td>Character</td>
<td>9</td>
<td>Name of the macro that must be executed after the &quot;user-written&quot; macro.</td>
</tr>
</tbody>
</table>

The MACRO column forms the primary key of the PMacro table. For example, the meta data of the PMacro table looks like this:

```
PMacro table
<table>
<thead>
<tr>
<th>MACRO</th>
<th>PROPERTY</th>
<th>PREMACRO</th>
<th>PSTMACRO</th>
</tr>
</thead>
<tbody>
<tr>
<td>%same</td>
<td>%prevar</td>
<td>%postvar</td>
<td></td>
</tr>
<tr>
<td>%previous</td>
<td>%prevar</td>
<td>%postvar</td>
<td></td>
</tr>
</tbody>
</table>
```

The %same "user-written" macro specified in the MACRO column must be executed inside of data step (according to the I value in the PROPERTY column). The PREMACRO and PSTMACRO columns specify macros forming an environment for executing "user-written" macros. The values of the PREMACRO and PSTMACRO columns corresponding to the %same macro specify that %prevar and %postvar macros form an environment for executing the %same macro. Such simple environmental macros are written by the programmer. The following programs present examples of such macros.

```
/*
PROGRAM PREVAR
DESCRIPTION Generates length and retain statements for variable corresponding to the specified column.
USAGE %prevar(varname);
*/

PARAMETERS varname - is the name of the table column.
/*
%macro prevar (varname);
%let prevar = &varname;
%if %length(&prevar) > 8 %then
  %let prevar = %substr(&prevar,1,8);
  length &prevar $ 200;
  retain &prevar "&mis";
%mend prevar;
*/

PROGRAM POSTVAR.
DESCRIPTION Assigns stored values for specified column.
USAGE %postvar(varname);
PARAMETERS varname - is the name of the table column.
/*
%macro postvar(varname);
%let prevar = &varname;
%if %length(&prevar) > 8 %then
  %let prevar = %substr(&prevar,1,8);
  &prevar = &varname;
%mend postvar;
*/
```

HOW TO DEFINE DATA AUTHORIZATION ACCESS, AND HOW IT WORKS

To illustrate the process of organizing an application, we consider the application supporting clinical experiments. We can specify two main tasks of this application: the first of them is an experiment's data acquisition, and the second is analysis of the acquired data. We consider these two tasks as independent applications. We can anticipate three types of users of these applications. They are: manager, pharmacist and physician. For each of them, we need to define authorized application objects and operations. The Figure 5 demonstrates how to define in the data dictionary tables an application, along with users' interactions with the applications' objects.

In order to implement data authorization access, the programmer has to solve the following general problems:
1. implement access to the application objects and operations
2. implement the mode mechanism.

In order to grant or deny users' access to the application objects and operations, the programmer need to develop an SCL program. Such a program must get user identification name and password and perform the following steps:
1. To retrieve from the AUserpas table the row with the USERID and USERPASW columns' values equal to the user identification name and user's password, and get from this row the USERGRP column value.
2. To select from the AUsermod table rows containing the same values of the USERGRP column received at the previous step. The retrieved rows constitute the
Applications Development

temporary table, with the same structure as the AUsermod table.

3. To merge the table produced in the previous step and the AAppmode table. The result is a new table with a list of applications and their modes authorizes for the user.

4. To provide for the user the list of applications received in the previous step, and to get the application password from the user, to identify with which application the user wants to interact. To produce table containing the selected application and its modes

5. To retrieve from the AModeobj and AModeopr tables rows matching values of the MODEID column with the table produced in the previous step.

Figure 5. Definition of data authorization access to application objects

To implement the mode mechanism the programmer needs to write macro, which gets mode definitions from the data dictionary tables. As a result, this macro creates a new table containing the list of modes authorized for the current user. This new table can also be presented to the user for selecting desired modes. The programs performing data authorization access are available from the authors.

SUMMARY

This paper has described the time-saving strategies and techniques for designing, developing and maintaining complex SAS applications. This paper has given the programmer some tips on how to write "user-written" macros performing data manipulation and data analysis.

The authors may be reached at the following address:

Samuel Berestizhevsky and Tanya Kolosova  
P.O. Box 1169, Nazareth-Elitz 17100, Israel  
Phone/Fax: +972 - 6554254  
Email: sasconsl@actcom.co.il

REFERENCES


SAS® is registered trademark of SAS Institute Inc. in the USA and other countries. © indicates USA registration.