Basic Concepts

of

SATAN

Analysis Features

February 2002

Edited by Karl-Heinz Schmidt (K.H.Schmidt@gsi.de)

CONTENTS

BASIC CONCEPTS OF SATAN ANALYSIS FEATURES

(From "Introduction and concepts of the GSI Experimental Data Acquisition and Analysis System" by F. Busch, D. Croome, H. Göringer, V. Hartmann, J. Lowsky, D. Marinescu, M. Richter, K. Winkelmann (ISSN 0171-4546), updated by R. Brauchle, R. Choc, Th. Fiebig, A. Grewe, K.-H. Schmidt, St. Steinhäuser, February 2002)

1. Introduction

SATAN is a software package to facilitate the on-line and off-line data processing associated with multi-parameter experiments. It allows an experimentalist to solve standard problems (e.g. spectrum evaluation, display, plotting, fitting) using a command language which enables manipulation of complex data structures and definition of non standard data processing functions in a high-level language.

The "Experimental Data Acquisition and Analysis System" EDAS[1] was created to acquire and analyze data collected in experiments carried out at the heavy-ion accelerator UNILAC. It has been available since 1975 for evaluating experiments at GSI. EDAS has undergone constant development, and many enhancements have been added to a great part by the users. While the EDAS system was originally developed on PDP-11's and a central IBM 3081 computer, the updated version of the SATAN subsystem has been ported to Dec VMS and Microsoft WINDOWS operating systems.

1.1 The components of SATAN

SATAN consists of the following:
- a command language,
- a high-level, experiment-oriented programming language,
- a set of support libraries for macros and programs,
- data libraries for raw data, analyzers (spectra), global parameters and pictures,
- program development support and command lists,
- interactive help and tutorial facilities, and
- documentation support.

The system may be used to control an experiment and to replay experimental data.

1.2 Data acquisition

Data acquisition has the following functions:
- Conversion of the analog signals created from physical detectors into digital numbers and
- Accumulation of these data and storage onto magnetic tape or disk

Data acquisition is performed with a specific software running on a dedicated front-end computer, apart from SATAN. (At GSI this is the Multi-Branch System.) Data can be send to the Remote Event Sever, running on another computer, via the local net There, the data are made accessible for on-line analysis with dedicated analysis programs, imbedded in SATAN, PAW, ROOT, etc., running on any computer that is connected to the net.

[1] R. Chestnut, B. Grasmueck, R. Hadsell, E. Krieger, W: Lebershausen, J. Lowsky, W: Plappert, M. Richter, H. P. Rother, O. Siart, R. Thieme, "The GSI data acquisition system (EDAS)", IEEE Transactions on Nuclear Science, Vol. NS-26, No. 4, August 1979

1.3 Data types

SATAN recognizes two distinct data types:

- list mode data (event by event) and
- spectra (accumulated events).

The list mode data consist of sequences of events. In SATAN terminology, an event represents the set of numerical values associated with a "physical event"; it is a collection of all parameters measured in connection with the "physical event".

An event consists of several parameters correlated in time; each parameter is a numerical value resulting after conversion of the analog signal captured by a detector. The parameters may represent physical coincidences (e.g. two detectors at different angles looking at coincident particles) or just different attributes of a single physical event (energy loss and residual energy of particles in a telescope experiment for example). To keep track of the correlations between the different parameters for later analysis, the are normally written even-by-event onto a storage device.

Spectra are accumulated arrays (one- or two-dimensional) where the events are sorted corresponding to their pulse height from some selected detectors.

1.4 Data flow

In SATAN the data flow is organized in the following way:

- Input and output of raw data, where the source and the target depend on the mode of operation,
- dispatching (sorting) of the raw data with respect to the desired events and forwarding them to the corresponding user-supplied analysis routine(s),
- processing of the sorted data to obtain results (e.g. generation of spectra), and
- user interaction with the system by means of commands for data-flow control, parameter modifications and display of the results.

The data flow in the real time (on-line) mode, where the source of the data is the experiment and in the replay mode, where the data are read from a data-set, is shown in Figure 1.

1.5 Data-analysis program

During an experiment the measured data must be controlled. This can be done by displaying single parameters or correlations of them. To monitor the experiment, it is often necessary to derive parameters of physical significance from directly measured parameters. This is the function of the analysis program in the real-time mode.

The complete evaluation of events is performed in the replay mode. The analysis program contains the experiment-dependent logic of event processing.

The results are stored in data structures ("analyzers"). The graphic representation of these data plays a key role in SATAN. By beams of a graphic display one- and two-dimensional spectra can be viewed either statically or "live". The "live display" provides a way of following data accumulation event by event.

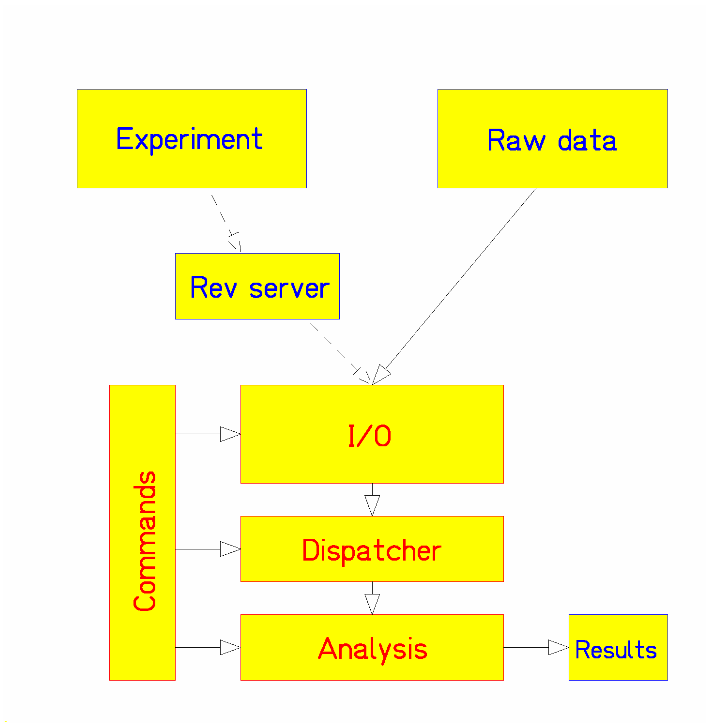

Figure 1: Data flow in SATAN, including front-end processing. The flow of data of several execution modes is illustrated. Data may originate directly from the experiment (real-time mode, dashed lines). In this case, the front-end system (apart from SATAN) copies immediately the measured raw data to a storage device (tape or disk) and simultaneously makes them available to SATAN via the Remote Event Server. Alternatively, the data may be fed in from disk or tape onto which they have been copied previously (replay mode). The dispatcher distributes the data to several analysis sections. The output may consist of modified raw data or spectra and can be stored also. All these processes are controlled interactively from the terminal via commands.

1.6 Command language

The interactive SATAN command language allows execution of standard functions. It gives the possibility

- to change the flow of control in analysis procedures,
- to change the flow of data by defining input and output data streams,
- to manipulate data,
- to display data,
- to fit model functions or
- to execute lists of commands.

The user communicates with the system by issuing commands and receives the results of the processing, mainly in a graphical form.

1.7 Programming language

The components of SATAN are written in the high-level programming language PL/I The implementation of SATAN on Microsoft WINDOWS uses VisualAge PL/I Professional, Version 2.17, IBM 2000. The user also writes his programs in PL/I using a set of macros, which can be expanded by the preprocessor to PL/I. All PL/I constructs are allowed in the user programs.

1.8 Macros and procedures

To comply with the variety of the experiments to be analyzed, SATAN is not a closed system, containing every conceivable analysis logic as a subset. Instead, the system calls subroutines written by the user according to well-defined rules in a high-level computer language. In these subroutines the user specifies the configuration and analysis logic of the experiment. All standards functions are supplied as commands, library routines or special language elements ("macros") by the system.

Such standard functions are

- input and output of raw data,
- spectrum accumulation,
- condition setting, and
- interactive graphics.

This modular construction makes the system very flexible, allowing the user to integrate new routines easily.

2. General ideas and concepts

2.1 Analyzer concept

The analyzer is a data structure and consists of
a spectrum of one or two dimensions,
conditions for accumulation of data, and
display windows and points.

The spectrum: The spectrum consists of a one- or two-dimensional data space with up to $2{\cdot}10^9$ bins in each dimension. (Of course, the number of bins is limited by the available memory of the computer.) The spectrum contents may be integer or floating-point numbers.

The conditions: A condition defines a spectrum segment for usage in analysis programs. It is set automatically for each event and may be used to branch depending on incoming data. It can be associated with an interval and be used to change the data flow of an analysis program depending on the input value. The condition can be changed interactively.

The display windows and points: These serve to store and display spectrum segments and channel numbers.

Analyzers are created in analysis programs by macros or interactively with commands. The attributes including name, spectrum limits and bin sizes may be changed interactively. The display and fit utilities operate with analyzers.

In most cases, the results of data analysis are spectra (analyzers). Analyzers can be dumped to datasets and fetched again into memory. By using the tools to structure directories and files, the user may build up a SATAN analyzer library.

2.2 System section and user section of SATAN

SATAN consists of the "system code", which performs standard functions requested by SATAN commands, and an optional "user section".

To provide more flexibility, SATAN was conceived as an open-ended system. This means that the user may add his or her own code in order to perform nonstandard functions specific to his own data processing. Since the user section and the system section are linked together into an executable module, the user section must be written carefully. An error in it might lead to overwriting the system code, leading to unpredictable results.

There are a number of ways to add user written code to SATAN:

- As a user **analysis procedure**. The rules are explained in detail in "Analysis programs and the analyzer concept". Basically, an analysis procedure has the aim to convert raw data input into modified raw data or into analyzers.
- As **user commands**. With a single keyword, the command name, optionally with additional parameters, the user may trigger execution of a complex processing.
- As **user exits**. The existing user exits are summarized in Table 1. They are invoked before the specified command is executed.

SATAN command	Procedure invoked	Function
LHALT	$MYHALT	stop data input
LINPUT	$MYINP	allocate and start input
LSTART	$MYSTART	start Monte-Carlo calculation
FMY	$MYFIT	user defined fit function
MYCOMMAND	$MYCOMMAND	user defined command

Table 1: SATAN user exits.

3. Data analysis

Assume a simple telescope experiment. A beam of particles hits a target. The reaction products create signals at the detectors which are related to energy and energy loss. The time-of-flight between the two detectors is measured, too. These three signals are led to digital converters to create at the output binary 16 bit numbers. A trigger pulse (either external or from one of the detectors) initiates the computer to read this set of three correlated parameters (this event) into the memory. From there they are copied onto an external medium (tape or disk).

Up to now, we do not need any analysis program: the directly measured data are immediately recorded. To monitor the experiment or to get numbers of physical relevance it is desirable for the experimentalist to calculate derived quantities during or immediately after the acquisition. In this case, the energy, mass and the atomic number of the scattered particle are derived from the directly measured quantities energy loss, time-of-flight, and energy. Windows are defined and, depending on whether a parameter hits the indicated interval or not (condition), spectra are accumulated. In this case: if the energy is in a certain range, a spectrum of mass and atomic number (nuclear charge) is accumulated. Event after event is analyzed in the way described.

Data structures are needed which contain the accumulated spectra and information about windows and conditions. For this purpose, SATAN provides analyzers.

3.1 Analyzer names

Analyzers are identified by names. An analyzer name is a sequence of alphanumeric characters (letters, digits or special characters "@" and "_"); the first character of the name must not be a digit. Analyzers may also be identified by a number which is derived from the sequence in which they are defined.

Analyzers are created in analysis routines using the macro AGEN or in sessions using the command AGEN. For example the command

```
AGEN REL_ENERGY
```

creates an analyzer with the name REL_ENERGY. Similar analyzers can be treated as arrays. Thus, the macro

```
AGEN(MASS(3));
```

creates three analyzers (MASS(1), MASS(2) and MASS(3)) in an analysis program.

3.2 Bin Size

To reduce spectrum extents, more than one channel number in any dimension may be defined to belong to one spectrum element ("bin"). The number of adjacent channels of one dimension contributing to a spectrum element is called "bin size". E.g.

```
AGEN(RE) BINS(16) LIMITS(0,511);
```

16 channels will be summed up to form one bin. This will make let your statistics look better and will save storage: in the example above the spectrum shrinks from 512 (for bin size 1) to 32 numbers. Note that all macros, commands and displays use the original channel numbers of the incoming data, regardless of the bin size. The bin size needs not to be integer. E.g.

```
AGEN (TAU) BINS(0.01) LIMITS(0,0.5);
```

creates the analyzer TAU with 50 bins.

3.3 Dimensionality and limits

Analyzers may have one or two dimensions. Lower and upper limit as well as bin size may be specified for each dimension.
A two-dimensional analyzer can handle several input values simultaneously:

```
AGEN (DE_E) LIMITS('0,10,0,191');
```

creates a two-dimensional analyzer DE_E where x runs from 0 to 10 and y from 0 to 191.

3.4 Conditions

A condition in the simplest case is a set of channel-number limits (one pair for each dimension) defined by the macro ACDEF or the commands ACDEF, W1SET (for one-dimensional analyzers) or W2SET (for two-dimensional analyzers). In the following example

```
AGEN (REL_ENERGY) LIMITS(0,1023) NCND(2);
ACDEF (REL_ENERGY,1,15,360);
ACDEF (REL_ENERGY,2,315,512);
```

the analyzer REL_ENERGY is created with two conditions. The subsequent invocations of ACDEF set the condition limits to run from 15 to 360 for condition number 1 and 315 to 512 for condition number 2. If the analyzer input values for an event lie inside the condition limits, the associated flag is set (in the macro ANAL). It can be checked in subsequent program sections (using the macro AC). Conditions are numbered sequentially starting from 1. It is also possible to access a condition by name, if the name is attributed by the W1DEF and W2DEF commands, respectively. All conditions must lie within the analyzer limits.

The concept of this simple rectangular condition is extended to a more general one for two-dimensional analyzers allowing any shape of the condition, defined by a closed polygon in the x-y coordinate plane.

3.5 Title and captions of the axes

For improving the graphic presentation of the analyzer, a character string appearing as a title and other strings for the captions of the axes may be specified by dedicated keywords (TITLE, CXASIS, CYAXIS) in the AGEN macro or in the AGEN command. As a default, the analyzer name appears as a title. The macro

```
        AGEN(DE)  LIMITS(0,250)  CXAXIS('Delta-E / arbitrary units') CYAXIS('counts');
```

defines the captions of the axes.

3.6 Type and mode

The analyzer type and mode specifies what kind of spectrum is defined for the analyzer.

The analyzer type relates to the contents of the analyzer channels. You can choose between 32-bits integer (FIXED) and 32-bits floating-point (FLOAT) spectra. The macro

```
        AGEN(XS) LIMITS(1,300) TYPE(FLOAT);
```

creates the analyzer XS, where the content of the analyzer channels is presented by 32-bits floating-point (DEC FLOAT(6)) numbers. Default type of analyzers created by the macro AGEN is FIXED. Since this type secures exact accumulation of integer contents up to 2.147E9, it is best suited for accumulating list-mode data. Default type of analyzers created by the command AGEN is FLOAT, because several commands may produce non-integer results which can only properly be stored in a FLOAT analyzer.

The analyzer mode relates to the nature of the quantity which corresponds to the analyzer channel. Two analyzer modes are supported, ANALOG and DIGITAL. An analog analyzer stores the values of a continuously varying quantity, e.g. an angular distribution. The number which characterizes a channel is the lower limit of the range ("bin") of one interval. A digital analyzer stores the values of a discretely varying quantity, e.g. the atomic number of nuclei. In this case, the number which characterizes a channel corresponds directly to the value of the discrete quantity. While the data of analog analyzers are presented by histograms, the data of digital analyzers are presented by symbols and/or by polygon lines. The result of several operations with analyzers, e.g. fit results, depend on the mode of the analyzer.

3.7 Macros, procedures and commands

Processing of an event in relation to an analyzer is performed by the macro ANAL. Each input value for an analyzer is compared with the condition limits and corresponding flags are eventually set. Then, if the input value is within the analyzers's limits, the corresponding spectrum bin is incremented by 1 (or by any specified amount).

A number of macros, procedures and commands is available for the user for various analyzer operations. The most important functions are summarized in the following:

- creation of analyzers, definition and modification of their attributes (macro AGEN, commands AGEN and AMOD),
- display of spectra and window limits (commands GDISP, GOVER, W1DISP, W2DISP),

- listing of analyzer spectra (command ALIST), analyzer attributes (AATT), moments (ASUM), and allocated space (ASPACE),
- setting of condition limits and windows (macro ACDEF, commands ACDEF, W1SET and W2SET),
- processing of a parameter in an analyzer, i.e. checking of windows and accumulation of a spectrum (macro ANAL), retrieving data from an analyzer (macro AGET),
- branching in the program, dependent upon results of a window check (macro AC),
- manipulation of spectra: zeroing, adding, subtracting, etc. (commands ACLEAR, ASET, AOPER, ADIFF, AINT, ACOPY, APROJECT, ACONVERT, APOLISH, AIMAGE, AMIRROR, AMOVE, ASHIFT, ANORM),
- complex operations like peak search in one dimension (command AFPEAK) and in two dimensions (command W2SET), deconvolution (command ADECON) and deducing correlations from two-dimensional spectra (command ARIDGE),
- application of the fit package (see separate documentation),
- saving analyzers on disk data sets and retrieving them (command ADUMP, AFETCH).

3.8. Structure of analysis procedures

The stream of experimental list-mode raw data is passed on for further processing in an event-by-event sequence to the analysis program. The user has to provide a routine to determine all operations to be performed on the event-parameter values (also called input values) of any single event, as for example window checking, numerical transformations, spectrum accumulation etc.

The analysis routines must be external PL/I procedures written in a standard form which is facilitated by use of macros. The following picture shows the skeleton of an analysis routine.

```
LISTPROC;
   .
   ... definition part
   .
ANTRY; /* begin of analysis part */
    EVENT DATA( event parameters );
       .
       ... analysis
       .
    ENDEVT;
ENDANL;
```

Figure 2: Skeleton of an analysis program

Beginning and end of the analysis routine are marked by LISTPROC and ENDANL, respectively. The first part of the procedure (“definition part”) contains definitions, initializations and storage allocations of all items which are used in the following analysis part, which starts with ANTRY. Analyzers have to be created here, global parameters declared a.s.o. This part is executed only once during the initial load of the SATAN system, whereas the analysis part is executed each time a block of event data is ready for processing.

For SATAN there is a loop to handle data event by event, since each block of event data consists of several events, each event being a group of list-mode parameter values. This loop is delimited by EVENT and ENDEVT.

The listing of Figure 3 shows a simple program written for the analysis of list-mode data in SATAN. You recognize the analysis part formed by the ANTRY and ENDANL. LISTPROC is used to provide the proper entry point for list-mode data.

Three analyzers AL, BETA, and GAMMA are created, AL with the channel limits 0 to 1023. The analyzer type is defaulted to FIXED, which means 4 bytes integers. The analyzers will contain spectra and control information. For example: analyzer BETA consists of a spectrum running from channel number 0 to 2000 and has one condition, namely number 1, which is set by ACDEF from 150 to 600.

Each event, which in this case consists of four parameters, is processed in the loop from EVENT to ENDEVT. Variable names are assigned to the event parameters (PATT, PA, PB, PC in this case) for subsequent reference in the program. For each event in the loop all condition flags of all analyzers will be cleared first.

```
 1    LISTPROC;
 2      AGEN(AL) LIMITS(0,1023);
 3      AGEN(BETA) LIMITS(0,2009) NCND(1);
 4      AGEN(GAMMA) LIMITS(0,100);
 5      /* end of analyzer definition */
 6      ACDEF(BETA,1,150,600);
 7    ANTRY;
 8      EVENT DATA(PATT,PA,PB,PC);
 9        ANAL(AL,PA);
10        ANAL(BETA,PB);
11        IF AC(BETA,1) THEN
12          ANAL(GAMMA,PC);
13      ENDEVT;
14    ENDANL;
```

Figure 3: Analysis program with basic macros: This simple SATAN program illustrates the creation of analyzers (AGEN) and the definition of a condition (ACDEF) in the definition part. In the analysis part, events are read in (EVENT), analysis and accumulation (ANAL), and condition-flag checking (AC) take place. The line numbers are only given for orientation.

Analyzer activity in the analysis part is invoked by the macro ANAL with the analyzer name and event parameters (input values) as arguments. If for any event this macro is encountered, the input values are compared with the condition limits. The corresponding condition bit is set. Further, the bin belonging to the input value (channel number) is incremented. That is what happens in lines 9 and 10.

The macro AC is provided in order to check the specified condition flag in subsequent program sections. In this example, AC is used to check the first condition flag of analyzer BETA whether it was set in the previous ANAL macro in line 10. That means, if the input value PB lies in between the channels 150 and 600, then and only then analyzer GAMMA is incremented.

After reaching ENDEVT, the process will continue at line 8 (EVENT) and the next event will be processed as long as there are events in the buffer. ENDANL in SATAN marks the end of processing one data buffer, and control is returned to the calling SATAN system routine which will provide for the next buffer.

The following example (Figure 4) shows a simplified evaluation of a telescope experiment. A SATAN analysis program is presented. It consists of macros , now interspersed with PL/I, the language into which the macros will be expanded .

```
LISTPROC;

  DCL(TZERO,EZERO,DEZERO) DEC FLOAT(6) INIT(0.0) STATIC,
      I BIN FIXED(15) STATIC,
     (MCAL,Z2,ECAL,TFAC,TCAL,EFAC,DECAL,DEFAC) DEC FLOAT(6) STATIC;

  AGEN(DENERGY) LIMITS(0,100) BINS(2) TYPE(FIXED);
  AGEN(ENERGY) LIMITS(0,100) BINS(2) TYPE(FIXED);
  AGEN(TIME) LIMITS(0,100) BINS(2) TYPE(FIXED);
  AGEN(MASSZ(3)) LIMITS(0,100,0,100) BINS(4) TYPE(FIXED);
  AGEN(CENERGY) LIMITS(0,3000) BINS(1) TYPE(FLOAT) NCND(3);

  ACDEF(CENERGY,1,21,23);                        /* energy conditions */
  ACDEF(CENERGY,2,201,233);
  ACDEF(CENERGY,3,2000,3000);

  LIST('ENTER CALIBRATION DATA');

  CALL $PROMPT(' ENERGY FACTOR',EFAC);/* read some calibration factors */
  CALL $PROMPT(' TIME FACTOR',TFAC);  /* from the terminal             */
  CALL $PROMPT('ENERGY LOSS',DFAC);   /* using system prompting routines */

  ANTRY;

    EVENT DATA(DE,T,E);                /* accumulate from the   */
      ANAL(DENERGY,DE);                /* detectors of the experiment */
      ANAL(ENERGY,E);
      ANAL(TIME,T);

      ECAL=EFAC*(E-EZERO);   /* calibrate energy and energy loss */
      DECAL=DEFAC*(DE-DEZERO);
      ANAL(CENERY,ECAL);     /* store calibrated energy */
      TCAL=TFAC*(T-TZERO);   /* calibrate time */

      MCAL=ECAL*TCAL**2;             /* evaluate the mass and */
      Z2 = (ECAL/MCAL) *DECAL*EFAC;  /* square of charge      */

      DO I = 1 TO 3;
        IF AC(CENERGY,I) THEN    /* for three energies accumulate */
        ANAL(MASSZ(I),MCAL,Z2);  /* mass and square of charge */
      END;

    ENDEVT;              /* end of event loop */
  ENDANL;                /* end of analysis   */
```

Figure 4: SATAN analysis program for telescope experiment.

The header and the generation of analyzers (macros LISTPROC and AGEN) are known already from the previous figure. Very often the problem occurs to read in data from the terminal. For this purpose, it is recommended to use the prompting routines offered by the SATAN system: a text is offered and the value is read in like in this example for the various calibration data. $PROMPT and $PROMPTO print a text and read a variable. All branching depending on the data type of the requested variable and all size checking and reprompting in the case of erroneous input is properly handled.

Three parameters are read in with the EVENT macro: DE (for energy loss), T (for time of flight) and E (for energy). These raw data are analyzed in the analyzers DENERGY, ENERGY and TIME. Now derived quantities (the calibrated data) are calculated. The calibrated energy is fed into the analyzer CENERGY, and as a result of some condition flag checking the analyzer MASSZ(i) are accumulated: mass and the square of the charge are kept for three different energy ranges.

Note that according to the requirement in this application all variables are declared static. Their values are kept from one call to another.

Having completed the analysis program, it has to be compiled (DOS command PLI) , linked (DOS command SLINK) and loaded with the system (by entering the name of the file of your analysis program in a DOS window). Data input is then started with the SATAN command INPUT.

4. Special options in analysis routines

4.1 Executing commands in procedures

SATAN provides the possibility to invoke commands from a procedure by the macro EXCMD. Figure 5 gives an example. At the very beginning of the analysis the display command is invoked. Then the display window is stored. After that at every 10,000th event the spectrum ETOT is drawn on the graphic screen without erasing the previous picture.

Note that the argument of EXCMD is a character string of variable length which can be assembled in the program:

```
        EXCMD('DISP AN('||IANL||')');
```

An element of the analyzer array AN will be displayed, the index IANL might have been calculated in a previous program section.

4.2 Data reduction

Often it is useful to cut the process of the analysis in several pieces. The experimentalist might be interested in derived quantities, which he wants to evaluate later, or a calibration of the input parameters has to be done, which produces a new set of parameters which forms a new event. In such a process some events might be selected as being no longer interesting. SATAN provides for that purpose a system routine which writes the contents of an array containing a modified event to an external device, which has to be allocated previously by the command “IOUTPUT”. The data written that way can serve without further processing as data input for another analysis program.

```
LISTPROC;
AGEN(ETOT) LIMITS(0,511);
AGEN(AN(8)) LIMITS(-20,100);
...
DCL IC BIN FIXED(15) INIT(-1);
...
ANTRY;
EVENT DATA(P,A(9));
  ...
  ANAL(ETOT,A(6));
  IF IC = -1 THEN DO;
    EXCMD('GDISP ETOT / YMAX(10000) YLOG');
    EXCMD('GSWINDOW / LAST');
  END;
  ...
  IC = IC + 1;
  IF IC = 10000 THEN DO;
    EXCDM('GOVER ETOT / WIN YLOG');
    IC = 0;
  END;
  ...
ENDEVT;
ENDANL;
```

Figure 5: Executing commands within a procedure: Every 10,000th event the analyzer ETOT is drawn on the graphic screen.

In the example of Figure 6 the input event is scanned: if the first parameter is larger than zero, parameters 3 and 4 will be written, further if the 8th parameter is in the range between 1100 and 1456, the parameters 10 to 16 are modified and written to the external device, too.

The allocation of the data set is done via execution of a command in the analysis program using the macro EXCMD. The necessary information to write a data-block header (like run, experimenter's name, and a comment) as well as the name of the data set are given. Because the macro EXCMD is located in the initialization part of the analysis program (before ANTRY), it is executed only once. Of course, the command could have been invoked also in the session. Then it should have been given before data input is started via the INPUT command.

4.3 Accessing analyzer data in user programs

Analyzer data may be used in an analysis program for calibration or other purposes. The macro AGET allows to read the content of a channel.

In the following, the content of channel three of analyzer AXY is written to the variable R_VAL:

```
R_VAL = AGET(AXY,3);
```

If an analyzer contains important data which should be kept during the analysis session, it may be protected against the command ACLEAR by the option PROTECTED of the macro AGEN.

```
LISTPROC;
...
AGEN(B) LIMITS(0,2047) NCND(1);
ACDEF(B,1,1100,1456);
DCL (I,IMODEVT(10)) BIN FIXED(15),
    (RFAC,OFFSET) DEC FLOAT(6);
EXCMD('IOUTPUT / DSN(EXAMPLE) RUN(R03) EXPERIMENT(MEIER) '||
   'COMMENT(''Compressed data'')');
ANTRY;
  EVENT DATA(P,PAR(16));
    ANAL(B,PAR(8));
    ... evaluate RFAC and OFFSET ...
    IF B > 0 THEN DO;
      IMODEVT(1) = 3; /* length of the event */
      IMODEVT(2) = PAR(3);
      IMODEVT(3)= PAR(4);
      IF AC(B,1) THEN DO ;
        IMODEVT(1) = 10; /* correct length */
        DO I = 1 TO 7;
          IMODEVT(3+1) = PAR(9+I)*RFAC + OFFSET;
        END;
        /* write to the output file */
        LOUT DATA(IMODEVT) N(10);
      END;
      ELSE LOUT DATA(IMODEVT) N(3);
    END;
  ENDEVT;
ENDANL;
```

Figure 6: Writing modified list mode data: The execution of the command IOUTPUT establishes the connection to the output file. Further a header for the data to be written is prepared. Events are read in. Depending on the value of the parameters, modified list mode data are written to disk by the SATAN macro LOUT.

4.4 Live display

SATAN offers a way to observe the events like they come in. The incoming event accumulates a one-dimensional spectrum drawn on the screen or displays two-dimensional correlations by marking the input value pair in a plane.

In the following the use of the live display is described.

```
LISTPROC;
...
AGEN(A) LIMITS(1,1024) BINS(2) TYPE(FIXED);
AGEN(A#B) LIMITS(1,1024,1,1024) BINS(10) TYPE(FIXED);
AGEN(A#C) LIMITS(1,1024,1,1024) BINS( 5) TYPE(FIXED);
ANTRY;
EVENT DATA(U,V,W,X,Z);
  DYNANAL;
  ...
  ANAL(A,V);
  ANAL(A#B,V,W);
  ANAL(A#C,V,X);
  ...
ENDANL;
ENDEVT;
```

The following command sequence has to be given in the session:

```
GSET / ZONE(3,1)
GDISP A / LIVE
GOVER A#B / LIVE
GOVER A#C / LIVE
INPUT / DSN(R70F5.LMD)
```

Three pictures on the screen are initialized. If during analysis an event satisfies the input condition of analyzer A, the spectrum “grows” on the screen according to the accumulated event. If the event satisfies the input condition of analyzer A and B or A and C, the corresponding parameters are drawn as a dot in picture 1 (or picture 2) on the screen. The resolution of a two-dimensional live display corresponds to the original quantities, only limited by the screen resolution. It is independent of the bin size of the analyzer.

4.5 Dynamic analysis

New analyzers to be filled by list-mode data may be created interactively by the command ANAL. All incoming list-mode data and all variables sorted into permanent analyzers in the existing analysis program can be used as input data.

4.6Monte-Carlo calculations

With a few modifications, the same tools, developed to write a routine for analyzing list-mode data can be used to write a Monte-Carlo simulation code which immediately fills analyzers or generates list-mode data. The routine given in the following example fills the analyzer YZ with a Gaussian distribution centered at the value of the interactive parameter ZP.

```
Listproc; /* This part is excecuted when starting SATAN */
  List('Sample for a Monte Carlo calculation');
/* Internal variables */
  Declare  Z Dec Float(6),
           IZ Bin Fixed(31);
/* Input parameters: */
  Pardcl(ZP) Init(92) Comment('Input: Nuclear charge of projectile');
/* Calculated spectra: */
  Agen(YZ)                 Limits(0,92) Bins(1)
                           Title(‘Z-Yields’)
                           Cxaxis(‘Z’)
                           Cyaxis(‘Y’)
                           Linesymbol(‘LT11’)
                           Type(Float);

Start; /* This part is excecuted by the command START */
  Loop; /* Event loop */
    Dynanal;
    Z = PGauss(ZP,7.3E0);
    IZ = FIXED(Z+0.5,31);  /* Round to integer values */
    ANAL(YZ,IZ);
  Next;
Endanl;
```

4.7 Space problems

SATAN adapts to the memory available in the actually used computer. This is the only limit imposed to the space of analyzers. Especially two-dimensional analyzers may eat up a considerable amount of memory. Eventually, an increased value of the bin size may serve to reduce the required space. Consider that the graphics screen does not resolve more than a few hundred channels in each dimension. The command ASPACE will report the space occupied by analyzers.

5. Interactive part of SATAN

5.1 Command syntax

A SATAN command consists of a command name followed by a list of parameters. Items in a parameter list must be separated by blanks. A slash "/" separates positional parameters from keywords. A command string may occupy several lines, up to 256 characters. The command is executed after entering the "Enter" key.

The basic syntax of a command is visualized below:

```
COMMAND  reqpos ...  optpos  ...  /  KEY(p)  ...

PARAMETERS
Reqpos        required positional parameter(s)
…
optpos        optional positional parameter(s)
…
Key(p)        optional parameter(s) identified by a keyword
…
```

5.2 Parameter types

Three different types of command parameters are possible.

- Required positional parameters
  These parameters are always required by the system and cannot be defaulted. They are entered as values (numbers, or character strings) immediately following the command name and are identified by the order in which they appear.
- Optional positional parameters
  They are also identified by the order in which they appear but may be omitted causing default values to take effect. Individual parameters may be defaulted by the use of adjacent commas; the list as a whole may be truncated from the end (right to left).
- Optional parameters
  Optional parameters are identified by a name (keyword) which may be followed by a value (number(s) or character string) enclosed in parentheses. Separated by the special character "/" from the list of positional parameters, they may be entered in any order. If omitted, default values take effect.
  Optional parameters without an argument act as a switch having the two possible values "0" and "1".

5.3 Rules for the use of SATAN commands

Most commands and parameter names may be abbreviated as long as the remainder is unique (either syntactically or per definition). Furthermore, in SATAN a command or a parameter can have more than one name (“aliases”).

If a parameter value includes delimiters (blank, comma or slash), it must be enclosed in single quotes.

Parameter names are converted to capital by the command decoder. To suppress this feature, the parameter name must me surrounded by “{}” brackets.

5.4 Functional classes

Generally the name of a command is chosen such that its purpose is more or less obvious. With few exceptions, the first letter of a command name denotes the functional class the command is belonging to:

A	Analyzer commands
C	Handling of commands
E	Handling of events
F	Fit commands
G	Graphics commands
I	Handling of interactive parameters
L	Input and output of list-mode data
O	Handling of command output
P	Handling of display points
S	Handling of command lists (SCOM lists)
W1	Handling of windows and conditions of one-dimensional analyzers
W2	Handling of windows and conditions of two-dimensional analyzers

5.5 Execution of commands

Besides the major application of using a single command in a dialogue mode, in SATAN a sequence of commands may be executed by interactively using a command list. In both cases the syntax of the used commands is the same. However, some important aspects have to be considered.

Command lists: A command list contained in a data set is invoked by the command EXEC. It may contain symbolic parameters. DO loops, IF and SELECT clauses are supported. Additionally, labels can be specified where the command list may be started and where the user is asked for continuation. Prompting is still done via the terminal. (See separate documentation.)

5.6 Global parameters

In most analysis programs, calibrations or other numerical calculations are to be performed with the raw data. This implies the use of numerical constants, the exact values of which are mostly unknown when the program is written. Therefore it is desirable that these constants can easily be modified during the analysis process by entering an appropriate command. Such constants shall be called “global parameters”.

SATAN supplies the macro PARDCL and the command IPAR by which global parameters can be defined, accessed, modified and listed.

5.7 Analyzer control commands

Generally analyzers are processed by a user-written program; however, the analysis is touched by some control commands changing analyzer attributes.

Most important are logical conditions defined and used in the analysis program. Their limits and internal numbers are part of the analyzer structure. Whenever an event is found within the limits of such a window, the corresponding condition flag in the analysis program is set to "on" and a user-defined action may take place. The actual limits of an analyzer condition are set by ACDEF or graphically by W1SET and W2SET, for one- and two-dimensional analyzers, respectively. Fetching conditions from a dump file by the command AFETCH / COND is another way to load predefined conditions into an analyzer.

Limits, bin sizes and number of conditions as well as type and mode of an analyzer are modified by AMOD. (The name may be altered too, but there is no effect on the analysis since analyzers are internally addressed by running numbers assigned in the sequence of creation.)

In addition to the macro AGEN, which generates analyzers within an analysis program, analyzers may be created dynamically by the command AGEN (and deleted again by ADES).

Analyzer attributes are listed by AATTR.

5.8. Spectrum handling

A set of commands affect the spectrum of an analyzer.

The whole spectrum is set to zero by ACLEAR. Data may be read from a file into a spectrum by AFETCH. Single channels or spectrum regions are modified by ASET. The command AOPER computes and stores the result of an arithmetic operation connecting two analyzers channel by channel. APROJECT performs the projection of a two-dimensional analyzer segment into a one-dimensional analyzer. The spectrum contents may be listed by ALIST.

Additionally, a number of display commands (GDISP, GOVER, GEXPAND, GXEXPAND, GYEXPAND) with a large variety of options is available to perform the graphical representation of spectra.

5.9 On-line analysis

On-line data acquired by the GSI MBS system[2] and distributed by the Remote Event Server[3] can be read by a dedicated client (winrevdlg.exe) on a WINDOWS/NT machine and written to local disk. SATAN is able to read these data simultaneously (command LINPUT dataset /CONNECT ONLINE) or in replay (command LINPUT dataset /ONLINE) in order to control the data acquisition. This task requires a user-supplied analysis procedure, adapted to the structure of the measured data.

[2] see http://www-gsi-vms.gsi.de/daq/home.html
[3] see http://www-aix.gsi.de/~goeri/mbsnew/revserver.html

5.10 Analyzer library

Analyzers can be saved to a file. Using system resources for the file-system management (e.g. the WINDOWS EXPLORER), analyzer libraries can be built up. By means of the command ADUMP, the whole analyzer structure is written to the analyzer library, spectrum as well as attributes. Thus also conditions etc. are stored and can be accessed again when the analyzer is recalled from the library by the command AFETCH, which creates analyzers or modifies attributes of existing analyzers if necessary. Items like spectrum, conditions or display windows and points may even be fetched singly.

5.11 Help system

Help information is available in SATAN by the HELP command. Detailed documentation on all SATAN commands and several general descriptions are available. Most part of the HELP system is written in HTML. Requiring help information by entering the HELP command will start the WEB browser and show the corresponding page of the HELP system.

The SATAN help system is also available in the WEB on the address
http://www-wnt.gsi.de/kschmidt/SATAN/graf.htm .

APPENDIX

About the SATAN/GRAF package:

The SATAN/GRAF package consists of two independently usable but closely connected parts. SATAN offers possibilities to read list-mode data (event-by-event data), to sort them into one- or two-dimensional spectra, called analyzers, and to perform complex operations with these analyzers. GRAF shows the graphic representation of analyzers or of numerical data which can be entered directly as one- or two-dimensional data arrays via an ASCII dataset.

The present document gives an overview on the analysis features of SATAN.

Features of the WINDOWS version of SATAN:

- Written in modern PL/I for Workstations.
- Multi-thread design.
- Elaborate support of GSI-type list-mode data structures with filter capability.
- Online usage by connection to Remote Event Server.
- Command language for batch mode.
- Numerous capabilities for manipulating analyzers.
- Powerful fit capabilities.
- Graphic presentation of 1-dimensional and 2-dimensional spectra.
- Automatic peak finding in one and two dimensions.
- Multi-spectra live display.
- Graphic package GRAF incorporated.
- Supports graphics output on plotters and to EPS or WMF files.

To run SATAN on WINDOWS, there are three possibilities

1. Users at GSI may test SATAN in a very simple way by executing this link: **P:\frstools\satan.bat**
2. It is more convenient to install SATAN on your own PC. You can do this also from outside GSI..
   See **http://www-wnt.gsi.de/kschmidt/SATAN-EXE.htm**
3. If all features of SATAN are required, including reading of list-mode data with a user-supplied analysis routine, a full installation which includes the PL/I compiler is required.
   See **http://www-wnt.gsi.de/kschmidt/SATAN/GHELP/install.htm**

This document is available in the WEB:

http://www-wnt.gsi.de/kschmidt/SATAN/GHELP/concepts/satan.pdf

For general information about SATAN/GRAF see
http://www-wnt.gsi.de/kschmidt/SATAN/graf.htm