391345

WSRC-TR-90-391

A Suite of RS/1 Procedures for Chemical Laboratory Statistical Quality Control and Shewhart Control Charting (U)

Kirk L. Shanahan

Westinghouse Savannah River Company
Savannah River Site
Aiken, SC 29808

SAFETY • RESPONSIBILITY • SECURITY
SRS
SAVANNAH RIVER SITE

PREPARED FOR THE U.S. DEPARTMENT OF ENERGY UNDER CONTRACT DE-AC09-89SR18035

WSRC-TR-90-391
UC-705

Keywords:
Quality Control

Retention Period:
Permanent

A Suite of RS/1 Procedures for Chemical Laboratory Statistical Quality Control and Shewhart Control Charting (U)

Kirk L. Shanahan

C. E. Coffey, Manager
Analytical Development Section

Publication Date: September, 1990

Westinghouse Savannah River Company
Savannah River Site
Aiken, S.C. 29808

ABSTRACT

A suite of RS/1 procedures for Shewhart control charting in chemical laboratories is described. The suite uses the RS series product QCA (Quality Control Analysis) for chart construction and analysis. The suite prompts users for data in a user friendly fashion and adds the data to or creates the control charts. All activities are time stamped. Facilities for generating monthly or contiguous time segment summary charts are included. The suite is currently in use at Westinghouse Savannah River Company.

CONTENTS

A Suite of RS/1 Procedures for Chemical Laboratory Statistical Quality Control and Shewhart Control Charting (U)

Introduction

This work was primarily motivated by the need for very strict process control of the Defense Waste Processing Facility (DWPF) process. The current process control scheme includes the use of laboratory data on feed stream composition directly in the process control function. This leads to the desire on the part of the chemical laboratory charged with the actual analysis to be able to provide accuracy and precision estimates on any routine sample analytical method. This in turn is a natural application of the field of Statistical Quality Control (SQC).

Further motivation for this work was provided by the requirements placed on the chemical laboratories here at Westinghouse Savannah River Company (WSRC) in the area of Quality Assurance (QA). QA requires complete documentation of quality control (QC) plans and implementations, and a well developed SQC program represents the best way to satisfy QA and QC requirements simultaneously.

Today, SQC (and the related area of Statistical Process Control (SPC)) are viewed by most businesses as competitive necessities. This is because the methodologies used in these fields have continually proven valuable. Because of this, software packages that implement parts or all of standard SQC/SPC techniques abound. This work has centered on the use of the software package called RS/1 and its options and related packages. This choice was dictated by already available hardware and software, and by the fact that the RS series of software offers distinct advantages in the areas of data analysis and sharing. These characteristics will become more important as the QC program progresses into the trouble-shooting phase.

Summary

A suite of ten modules has been constructed which customize standard RS/1 quality control packages for use at WSRC. This suite has been installed in the analytical chemistry laboratory facilities in the DWPF, the DWPF pilot plant facility (TNX), and the Savannah River Laboratory (SRL). The package is menu-driven and performs the following tasks:

- Prompts technicians for data entry
- Timestamps all activities
- Creates and/or modifies the Shewhart Control Charts and separate data tables

- Conducts statistical trend analysis

- Produces hardcopy output of charts and tables

- Produces summary charts and tables (primarily for QA)

The suite focuses on user-friendliness throughout, and attempts to automate the control charting process as fully as possible. In the DWPF application, data entry actually will occur on a Laboratory Information Management System computer, and automatic data transfer to the QC package has been demonstrated.

Most of the users of this package (technicians, chemists, and management) were unskilled in computers and/or control charting, so extensive training was undertaken. Additionally, a Quality Circle was formed to serve as a forum for discussion of operating experience with the package and related topics.

Future plans consist of continued training of the Quality Circle members in other types of control charting (CUSUM, EWMA, etc.), computer use and data sharing concepts, and general quality control. Possible formation of more localized Quality Circles is also being considered. Also, integration of this effort into site Total Quality plans is underway.

Background

Statistical Quality Control is a very old field. Original work in the area began in the 1920's, and manuals and books were published on it in the late 1940's and early 1950's. The technique of control charting however, was until more recently generally considered to be a Process rather than Quality control technique. With the advent of severe Japanese competition in the 1970's and 1980's, control charting became recognized as one part of a good laboratory quality control program.

The Shewhart control chart was the earliest variety. It has the advantage of simplicity. Data is plotted directly on a graph in the order it is generated, and is compared statistically to pre-defined acceptance limits. These statistical tests include searches for trends (such as linear drifts) in the data. When a statistically unlikely event is detected, a control flag is generated, and pre-defined control actions are taken to correct the problem.

Depending on the nature of the data and the acceptance limits, the Shewhart chart may not be sensitive to certain events. For this reason, other types of charts were proposed and implemented. The primary disadvantage of these charts is that in most cases the data is transformed mathematically before it is plotted. Thus, one loses the ability to look at the chart and intuitively understand it. In addition, the nature of the transformation can be quite complex, leading to a multitude of

charts being generated from one data set. Understanding and using these charts is usually beyond the ability and desire of most operators and technicians, and the latest successful applications of quality control almost always emphasize the involvement of the operator or technician in the quality process.

The RS software series, produced by BBN Software Products Corporation, is an integrated data management and analysis software package that consists of the core package RS/1, and several add-on options and expansions. These add-on options range from experimental design packages to serial data input packages to quality control and an expert system shell. The RS series software is primarily aimed at use on Digital Equipment Company computers (Vax, MicroVax), but major parts also run on IBM PC's under DOS. The software package described in this report specifically uses features of standard RS/1 and the QCA (Quality Control Analysis) optional package.

Overview of the Suite

The suite of software procedures consists of the following modules:

- MENU (top level procedure in typical application)
- MAKE_QC_CHART (primary procedure)
- ADD_POINTS_TO_QC_CHART (primary procedure)
- MAKE_TEMP_DATA (sub-procedure)
- BUILD_QC_CHARTS (sub-procedure)
- QC_Y_AXIS_SETUP (sub-procedure)
- QC_HARDCOPY (sub-procedure)
- IBMPC_FIXUP (sub-procedure)
- SUMMARY_QC_CHART (primary procedure)
- MULTIPLE_COMPONENT_QC_CHART (primary procedure)

Primary procedures are stand-alone procedures that require nc arguments in the call statement. Sub-procedures are called by the primary procedures and receive passed arguments.

MENU is a BBN supplied procedure for presenting a suite of procedures via a menu. The programmer fills in user instructions and the related procedure calls. The user simply calls the menu and then makes choices from it.

MAKE_QC_CHART (MQC) constructs the Shewhart charts from operator input once only, when the chart is first started. It also fills in customization parameters on the charts which stay with the chart from that time on.

ADD_POINTS_TO_QC_CHART (AQC) adds new data to existing charts.

MAKE_TEMP_DATA is a sub-procedure used by MQC and AQC for technician data entry and correction.

BUILD_QC_CHARTS (BQC) is the sub-procedure that takes the data and makes or modifies the charts.

QC_Y_AXIS_SETUP is a sub-procedure used by BQC that sets the low and high of the control chart's Y axis so that all data and messages will be seen.

QC_HARDCOPY is a sub-procedure that produces hardcopy output of the charts and associated data tables.

IBMPC_FIXUP is a sub-procedure that is used to temporarily bypass display problems encountered with the use of the free terminal emulator, RSTERM, which BBN supplies for use on IBM personal computers.

SUMMARY_QC_CHART is a procedure that extracts either a monthly summary or a contiguous data block summary, primarily for QA documentation purposes.

MULTIPLE_COMPONENT_QC_CHART is a procedure that is used when multiple component standards are used. It presents a more efficient data entry routine and does not require a graphics display of the Shewhart charts.

General Modes of Use

A typical user of this suite of procedures will use it in one of three general modes. First the user could be making or modifying single method/single component standard Shewhart charts. In this instance, the user (technician) has run a standard through an analytical method and produced a result (or results if the standard was run in duplicate or triplicate for example). The user then calls up the menu and selects either MAKE a chart or ADD TO a chart. The procedure prompts the user for the data and allows correction of typing mistakes, modifies or creates the chart(s) and data table, trend analyzes the data for statistically significant events (marking those for identification), asks the user if hardcopy is desired, makes hardcopy if requested, and returns the user to the menu. The key to this mode is that the analytical method is for single component standards. Multiple component standards are more efficiently handled by the second mode, described below. Data for a single run or multiple runs can be entered at one time.

The second use mode is for the case of multiple component standards. In this case the user has run a multiple component standard, perhaps in duplicate or triplicate, and has data for each component which must be charted separately. The user selects the MULTIPLE COMPONENT choice from the menu and the procedure asks the user for which multiple component method he will be entering data. (The information concerning the method is entered separately by a chemist or supervisor. See the instruction manual in the Appendix for more details.)

After selecting the method, the procedure prompts the user for the data and allows corrections if required. It then adds points to (or creates) the charts and tables, trend analyzes

the data, and checks for out-of-control indicators. It produces a report table which is displayed for the user. An empty report table indicates no out-of-control indicators, while entries for the data and rule violations are entered when an indicator is noted. With this mode, only single runs can be entered at a time.

The third use mode is generation of summary charts and tables. The presumption behind the suite is that all data for a method/component will be entered into a single Shewhart chart and data table and will be kept on-line for at least two years (five years is preferable). This necessitates the capability to produce summaries of regions of interest. These summaries are obtained by choosing that option from the menu. The procedure prompts the user for which method/component a summary is desired and for which type of chart is desired (a complex and busy form with lots of information or a cleaner form with a little less information). It then generates the appropriate chart and displays it. Upon confirmation it prints a hardcopy of the chart and the related data table (if requested). The procedure then asks if the user wishes another. When the user is finished, he is returned to the menu.

In some cases, laboratory management may wish to exercise more control over control limits. This software allows the definition of 'standard' methods. A 'standard' method in the context of this suite is a method for which certain information has been stored on-line. This information is accessed each time a technician enters data, and replaces actual technician entry. This information is also required to use the multiple component standard procedure. A full description of how to create a 'standard' method and allow it to be used for a multiple component method is included in Appendix III.

Specific Advantages of RS/1 as Applied to SQC

The primary advantage of using RS/1 as the package for daily quality control is data sharing and networkability. The suite of procedures stores data tables and charts in a 'grouphome'. A grouphome is a special area on the computer that can be accessed by anyone with the appropriate privileges. These privileges are assigned as part of the user's computer account. Thus the technicians, chemists, supervisors, and quality control personnel can share access to the quality control records on-line. In fact, with the use of wide-area and local-area networks, these people can be in physically different locations and be examining and discussing the exact same data, as soon as it is entered into the computer.

This is a very powerful advantage to quality control trouble-shooting. The experts in the method need not be physically present to examine data, and substantial time savings can be had by allowing the technician to call the expert directly instead of going through a circuitous chain of

command.

An additional important advantage to the grouphome method of data sharing for laboratory management should be mentioned. With good quality control records and programs, the laboratory is in an excellent position to answer questions immediately regarding the state of health of a particular method. This also is a time saver, in that time that would have been wasted digging information out of hardcopy records and notebooks is replaced by a quick logon and look-see. Further, performance reports can be generated automatically at appropriate time periods for inclusion in quality control reports, which can provide answers to such questions before they are even asked.

The latest versions of RS/1 have routines for data transfer between RS/1 and Lotus or SAS or even ASCII files. Thus the data can easily be transported to favorite software for more extensive analysis in familiar environments. However RS/1 itself has extensive statistical analysis capabilities and it is anticipated that analysis of the data within RS/1 will be the preferred operation mode. Also, the data tables can be used to create other standard control charts such as CUSUM or Exponentially Weighted Moving Average charts, which are also part of the RS/1 QCA optional package.

Software Quality Assurance

This document is intended to serve as the basis of any software quality assurance requirements that might be placed on the chemical laboratory with regards to the suite of procedures described herein. In order to do that, the procedures must be listed, documented and tested with documentation of the test results. This is done below.

Documented versions of the procedures are included in Appendix I. Noted in the comments are creation dates, installation dates, and dates of major modifications. Several of these are noted as trial installation of these procedures was instituted at TNX, with subsequent revision based on technician and other user feedback. It is expected that some customization of the code will occur in each area in order to accommodate local needs. All subsequent changes to the code should be documented in the code comments, with any major revision being reported separately.

Two sets of data were constructed to test the software. The first set consisted of 37 data points of the single determination type. The data set was constructed to illustrate a control flag for each of the ten rules. The data table is shown in Table 1, and the resultant control chart is shown in Figure AI-2. A total of fourteen flags are indicated. It is important to note that four of these (on subgroups 2, 4, 10, and 12) are included to illustrate that the "2 out of last 3" and "4 out of last 5" rules trigger as soon as the condition is satisfied, i.e., 3 and 5 points respectively are not required for a flag to be inserted.

The second set of data consists of 35 data points of the multiple determination type. Again the data set was constructed to illustrate one control flag for each rule, but in this case, one point was also constructed to test the standard deviation flag for exceeding the control limit. The data table is shown in Table 2, the means control chart is shown in Figure AI-3, and the standard deviation control chart in Figure AI-4. The RS/1 software calculates the mean and standard deviation from the raw data (shown in Table 2), and this calculation was verified by hand calculations.

Table 1. Test Data for Individuals Control Chart

	STDVALUE	RANGE % of STD	DATE	VALUE1
1	100	6	01-JAN-90	107
2	100	6	02-JAN-90	105
3	100	6	03-JAN-90	105
4	100	6	04-JAN-90	103
5	100	6	05-JAN-90	103
6	100	6	06-JAN-90	101
7	100	6	07-JAN-90	101
8	100	6	08-JAN-90	101
9	100	6	09-JAN-90	93
10	100	6	10-JAN-90	95
11	100	6	11-JAN-90	95
12	100	6	12-JAN-90	97
13	100	6	13-JAN-90	97
14	100	6	14-JAN-90	99
15	100	6	15-JAN-90	99
16	100	6	16-JAN-90	99
17	100	6	17-JAN-90	100
18	100	6	18-JAN-90	100
19	100	6	19-JAN-90	100
20	100	6	20-JAN-90	100
21	100	6	21-JAN-90	100
22	100	6	22-JAN-90	100
23	100	6	23-JAN-90	100
24	100	6	24-JAN-90	100
25	100	6	25-JAN-90	100
26	100	6	26-JAN-90	100
27	100	6	27-JAN-90	100
28	100	6	28-JAN-90	100
29	100	6	29-JAN-90	97
30	100	6	30-JAN-90	103
31	100	6	31-JAN-90	97
32	100	6	01-FEB-90	103
33	100	6	02-FEB-90	103
34	100	6	03-FEB-90	97
35	100	6	04-FEB-90	97
36	100	6	05-FEB-90	97

Table 2.
Test Data for Multiples Control Charts
(Including Calculated Means and Standard Deviations)

	MEAN	ST DEV	STD VALUE	RANGE % of STD	DATE	VALUE1	VALUE2
1	104.5	4.949747	100	6	01-JAN-90	108	101
2	104.0	1.414214	100	6	02-JAN-90	105	103
3	102.5	3.535534	100	6	03-JAN-90	105	100
4	101.5	2.121320	100	6	04-JAN-90	103	100
5	101.0	2.828427	100	6	05-JAN-90	103	99
6	100.5	0.707107	100	6	06-JAN-90	101	100
7	100.5	0.707107	100	6	07-JAN-90	101	100
8	100.5	0.707107	100	6	08-JAN-90	101	100
9	93.0	0.000000	100	6	09-JAN-90	93	93
10	96.0	1.414214	100	6	10-JAN-90	95	97
11	97.5	3.535534	100	6	11-JAN-90	95	100
12	98.5	2.121320	100	6	12-JAN-90	97	100
13	99.0	2.828427	100	6	13-JAN-90	97	101
14	99.5	0.707107	100	6	14-JAN-90	99	100
15	99.5	0.707107	100	6	15-JAN-90	99	100
16	99.5	0.707107	100	6	16-JAN-90	99	100
17	100.0	0.000000	100	6	17-JAN-90	100	100
18	100.0	0.000000	100	6	18-JAN-90	100	100
19	100.0	0.000000	100	6	19-JAN-90	100	100
20	100.0	0.000000	100	6	20-JAN-90	100	100
21	100.0	0.000000	100	6	21-JAN-90	100	100
22	100.0	0.000000	100	6	22-JAN-90	100	100
23	100.0	0.000000	100	6	23-JAN-90	100	100
24	100.0	0.000000	100	6	24-JAN-90	100	100
25	100.0	0.000000	100	6	25-JAN-90	100	100
26	100.0	0.000000	100	6	26-JAN-90	100	100
27	100.0	0.000000	100	6	27-JAN-90	100	100
28	98.0	0.000000	100	6	29-JAN-90	98	98
29	102.0	0.000000	100	6	30-JAN-90	102	102
30	102.0	0.000000	100	6	01-FEB-90	102	102
31	98.0	0.000000	100	6	02-FEB-90	98	98
32	98.0	0.000000	100	6	03-FEB-90	98	98
33	102.0	0.000000	100	6	04-FEB-90	102	102
34	98.0	0.000000	100	6	05-FEB-90	98	98
35	102.0	0.000000	100	6	06-FEB-90	102	102

This test scheme is adequate because of the modular design of this suite of procedures. BUILD_QC_CHARTS (BQC) is the primary workhorse for chart construction and is used by MAKE_QC_CHART (MQC), ADD_POINTS _TO_QC_CHART (AQC), and MULTIPLE_COMPONENT_ QC_CHART (MCC). Arguments passed with call statements are used as flags to tell BQC which primary procedure called it. The logic inside BQC then performs the appropriate tasks based on the calling routine and data type (individual or multiple determinations). MAKE_TEMP_DATA is used by MQC and AQC for manual data entry and correction. MCC performs that function itself, but uses BQC for the charting.

QC_HARDCOPY is a separate module because each computer has separate printing facilities. QC_HARDCOPY represents one of the major customization points in this suite. IBMPC_FIXUP likewise is hardware specific. SUMMARY_QC_CHART is essentially stand alone, except that it also uses QC_HARDCOPY for printing. These routines are verified by use. Hard copy produced by SUMMARY_QC_CHART is included in Appendix I for comparison.

One problem with the core RS/1 QCA procedure $MAKECHART (which constructs a new chart) was noted when only a single point was entered. The entry of just one point on a single chart is unusual, and the programmers apparently did not anticipate this case. $MAKECHART adds an additional row onto the Shewhart chart underlying graph table which is empty. The effect of this is to cause the trend rule analysis to assume there is one more point present than there really is. In certain instances, this can lead to an erroneous out-of-control indicator. Since this case is so unusual, an administrative solution to the problem should be implemented, rather than a software solution.

Other minor problems were noted during programming. Each of these was successfully addressed in the RPL code, and comments were added to explain the relevant sections of code.

Implementation

The implementation of this software should be a phased one. The first phase is characterized by using control limits based on expectations. After accumulating at least twenty to thirty data points, the first phase is completed by re-evaluating the control limits in light of actual performance. At this point the limits may have to be either lowered or raised, or may not need to be changed at all. This phase lasts until this re-evaluation occurs.

Now the second phase begins. Usually a method that has not been subject to a statistically based quality improvement program will show a non-random distribution of errors. The second phase of the quality control implementation is the elimination of 'special cause' events. 'Special cause' events are typically those events that produce 'flyer' data points. The objective of the second phase is to eliminate flyers,

thereby producing data that is randomly distributed. This phase lasts until the effort needed to eliminate special causes is invested, which can typically take from six to eighteen months per method.

Random distributions should be considered the norm, rather than the exception. When a method is in 'statistical control' it has a random distribution. This implies a certain number of data points will fall out of the control limits, and the occurrence of that event should be considered normal. Only when a disproportionately high number of such points occurs is a quality problem indicated.

Besides the simple 'out-of-limits' consideration, the RS/1 software tests for trends in the data by applying a total of 10 rules. These rules are listed in Table 3 below (the term 'sigma' refers to the standard deviation of the data). Rules 1 and 5 are the standard 'out-of-limits' rules. Rules 2,3,4,6,7, and 8 check for consistent drift in the data. Rules 9 and 10 check for stratification (the condition where the method jumps between two distinct sets of results randomly, or when there is insufficient variation in the data as implied by the control limits).

Table 3 also lists the occurrence probabilities for each event described in the rule. These probabilities were calculated by the method described in the AT&T Statistical Quality Control Handbook, pgs. 180-183. The cumulative probability of getting a rule violation is approximately .02, thus a natural frequency of about 2 out of 100 'out-of-control' indicators is expected. More than this would indicate a quality problem, either with the method or with the control limits of the chart (depends on the current phase of implementation).

The above rules should only be applied to randomly (normally) distributed data. When dealing with 'Multiples' type data, a standard deviation control chart is also produced. In this case however, only rules 1 and 5 are implemented, since variances (the square of the standard deviation) follow a chi-squared distribution. While occurrence probabilities for the other rules could be calculated, the interpretation of the event cause is extremely difficult.

The RS/1 procedures use a restricted set of rules to check the standard deviation chart. Since standard deviations are not randomly distributed, the use of trend rules based on random statistics is not advised. Instead, only rules 1 and 5 are used (the standard 'out-of-limits' rules). The restricted rules are stored in a table whose name is hardcoded into the source code (usually RESTRICTED_TREND_RULES). This table is a subset of the table used to do the trend rule checking on means and individuals data.

For completeness, a copy is shown in Table 4. The table holds RS/1 supplied procedure names ($QC_TREND1 and 5) that check for each rule's condition. The codes 'MA', MER', 'MES', etc.

refer to various kinds of charts that can be generated within the RS/1 QCA package. Column 7 ('SA') was added for use by these procedures.

The third implementation phase begins when a normal distribution of errors is achieved. Now the actual performance of the method can be evaluated versus quality goals. In light of available resources, methods that do not meet the quality goals are examined and improvements sought. Techniques used in this phase include designed experimentation, cause-and-effect diagramming, and others. This phase should be an on-going one.

Table 3. RS/1 Trend Rules

Rule No.	Rule Description	Occurrence Probability
1	1 point above 3 sigma	.00135
2	2 of 3 points above 2 sigma	.00151
3	4 of 5 points above 1 sigma	.00267
4	8 points above center line	.00391
5	1 point below -3 sigma	.00135
6	2 of 3 points below -2 sigma	.00151
7	4 of 5 points below -1 sigma	.00267
8	8 points below center line	.00391
9	15 points inside +/- 1 sigma	.00325
10	8 points outside +/- 1 sigma	.00010

Table 4. The Restricted Trend Rule Table Used on Standard Deviation Control Charts

0	1 Rule Number	2 MA	3 MER	4 MES	5 IA
1	1	$QC_TREND1	$QC_TREND1	$QC_TREND1	$QC_TREND1
2	5	$QC_TREND5	$QC_TREND5	$QC_TREND5	$QC_TREND5

0	6 IE	7 SA
1	$QC_TREND1	$QC_TREND1
2	$QC_TREND5	$QC_TREND5

APPENDIX I. Documented Source Code, Output, and Internal Tables

Table of Contents

PROCEDURE 1. #MENU (Supplied as part of QCA-I by BBN Software Products Corp.)

```
/* MENU is the top-level procedure for the menu-driven demo       */
/* which is included with the QCA software.  The whole collection */
/* of procedures shows how you can use the QCA procedures with    */
/* other RS/1 capabilities from within your own RPL procedures.   */
/* It is intended to be an example, rather than a part of the QCA */
/* system.  You can copy the QCA demo procedures into your own    */
/* procedures table by typing "CALL $COPY_QC_DEMO".               */

procedure(menutab);
    {
menutab='#qcmenu';
    firsttime = TRUE;

    do while TRUE;
        {
        if empty(menutab) then
            menutab=gettable("Name of menu table to use? ", , TRUE, "#QCMENU",
"Enter the name of a menu table of your own or use the default #QCMENU.");

       if not($iswatched(menutab)) then
           {
           if not(firsttime) then
               call erase(true);
           else firsttime = FALSE;
            watch col 0 of table(menutab) nocolnumbers noheader colwidth 80;
           }

        action = getcolresp("Select an activity (by number): ",
           FALSE, menutab, 0, EMPTY, EMPTY, TRUE, ,
"To choose an activity, enter the number of activity from the menu.
Just press RETURN to exit from this menu.");

        if empty(action) then
            return;

        act = row action col 1 of table(menutab);

        if empty(act) then
            donext;
       else if (caps(act) = "EXIT") then
           doexit;
        else
            exec(act, , status);
        }

    }
```

PROCEDURE 2. #MAKE_QC_CHART

```
/* This is a procedure to construct Individuals or Mean/St. Dev. QC charts. */
/*     Originator: Kirk L. Shanahan     Date: 8/14/89                       */
/*  Version 0   */
/*  Version 0.01  Fixed data correction bug.  9/8/89  KLS     */
/*                Added version printout      9/8/89  KLS     */
/*                Fixed up banner             9/8/89  KLS     */
/*  Version 1.0  Major revision - deleted grouphome subdirectories   */
/*                              - added user input of control limits */
/*                              - prettied up charts                 */
/*                              - added timestamp of all data        */
/*               reprogramming by Kirk Shanahan - start 9/19/89      */
/*              Major Revision - Began exclusive GROUPHOME utilization */
/*                              - split part of procedure off into subroutine */
/*                                called BUILD_QC_CHARTS (BQC)              */
/*                              - added a call flag so BQC would work with    */
/*                                MQC and AQC.                                */
/*               reprogramming by Kirk Shanahan - start 9/25/89      */
/*                                                                           */
/*  Version 2.0  Major Revision - Changed call to BUILD_QC_CHARTS by adding  */
/*                                DISPLAY_FLAG variable (set to 0) for       */
/*                                compatibility with MULTIPLE_COMPONENT_QC   */
/*                                _CHART procedure.                          */
/*                                Kirk Shanahan  3/1/90                      */
/*                                                                           */
/* This procedure constructs a Shewhart-type Control chart from either       */
/* individual or multiple determination data. It can use a list of 'standard'*/
/* tests that are stored in a CONTROL_LIMITS table, or it can construct a    */
/* stand-alone chart for 'non-standard' tests. The CONTROL_LIMITS table      */
/* the following information: Name of test, Control Limit (3 sigma as a      */
/* percent of the standard's value), Units of the test (Y axis label),       */
/* Table-type (individuals or multiples), a chart Title, and the number of   */
/* points per line when the data is of the multiples type.                   */
/*                                                                           */
/* The procedure prompts the user for the name of the test via a direct      */
/* question or via a listing of the standard tests, then prompts the user    */
/* for the appropriate data (done in subprocedure MAKE_TEMP_DATA). Next, it  */
/* copies the new data to the grouphome and calls the subprocedure (BUILD_   */
/* QC_CHARTS) which constructs the graphs and does standard Western Electric */
/* trend analysis, and finally asks the user if hardcopy output is desired   */
/* (achieved via subprocedure QC_HARDCOPY).                                  */
/*                                                                           */
/* The current implemetation of this procedure is via a menu system taken    */
/* from the QCA option to RS/1. The core chart building and trend analysis   */
/* procedures used in this procedure also came from the RS/1 QCA option.     */
/*                                                                           */
/*                                                                           */
/*  START OF PROCEDURE MAKE_QC_CHART. CURRENT VERSION 2.0                    */
/*                                                                           */
erase;
/*  Clear the screen  */
type "";
/* This construct types a blank line. It is used for spacing on the screen.  */
type "";
```

```
type "";
type "";
type "";
type "";
type "Welcome to the QC world. This is procedure MAKE_QC_CHART (MQC).";
type "MQC will prompt you for new data and create a new data table.";
type "MQC will then create the appropriate QC chart(s).";
type "";
type "  Version 2.0";
type "";
type "";
/* Banner - contains procedure name so user can verify he is in the right */
/*          one.                                                          */
if tableexists('temp_data') then delete temp_data;
/*  temp_data is the temporary data table that this procedure uses for    */
/*  data entry. It eventually is renamed to a permanent table in the      */
/*  GROUPHOME.                                                            */
getname:
/* This is a label used as a jump-in point from a later GOTO stmt.        */
/*  It is the starting point of the procedure.                            */
IF YESANSWER('Are you making a chart for a standard test?') THEN
/* If the user answers "Yes" here, the procedure goes to the CONTROL_LIMITS */
/* table and displays a list of standard tests for the user to choose from. */
/* A "No" sends the procedure to the section that gets a name directly from */
/* the user.                                                               */
 {
type"";
type "A list of methods will now be displayed.";
type "You should locate the test you want and remember its row number.";
type "You will be asked to enter the row number after the display is finished.";
type "If the test you want doesn't appear on the list, it is not a 'standard'";
type "test. In this case enter a 1 (chooses NON_STANDARD to go back to the start
.";
type "";
checker1: if not yesanswer("Are you ready?") then go to checker1;
/* The TYPE-CHECKER1 block above is meant to give the user a chance to get  */
/* for the upcoming display.                                               */
   get_test_name:
   DIS COL 1 OF #tmx_CONTROL_LIMITS;
/* The table in this statement is the CONTROL_LIMITS table. Here it is stored */
/* in a grouphome subdirectory and 'personalized' via the 'tmx' designation  */
/* for the Analytical Development Section QC effort.                       */
   control_row= GETNUMBER('Please choose which test by entering the number:');
/* The user enters the row number of the test for which a chart is to be   */
/* constructed.                                                            */
      if control_row=1 then goto getname;
/* The CONTROL_LIMIT table should always have row 1 col 1 be "NON_STANDARD". */
/* This allows the user the chance to go back to the beginning when needed  */
/* via the GOTO above.                                                     */
   dis row control_row of col 1 of #tmx_control_limits;
   if not yesanswer('Is this the test you wanted?') then goto get_test_name;
/* Display the user's choice and confirm it.                               */
   chart_name = row control_row col 1 of #tmx_control_limits;
   table_type = caps(row control_row col 4 of #tmx_control_limits);
/* Get the stored info CHART_NAME and TABLE_TYPE on the test from the CONTROL */
/* _LIMITS table.                                                          */
```

```
   std_flag=TRUE;
/* STD_FLAG is a flag used in the subroutines to tell if the test is a      */
/* 'standard' test.                                                         */
   goto check_group_table;
/* Jump past the next block because it is for a 'non-standard' test.        */
 }
std_flag=FALSE;
control_row=0;
/* Since this section is reached only for 'non-standard' tests, the flag    */
/* indicating a 'standard' test is set FALSE, and the CONTROL_ROW variable  */
/* is set to 0, so that it will not cause problems later if left empty.     */
chart_name =GETTEXT("What is the name of the QC Chart you wish to make? ");
/* Chart_name is the variable that holds the core of the table and QC chart */
/* names. GETTEXT is an RPL funtion for terminal I/O that gets a string.    */
/* Here the input string is assigned to 'chart_name'.                       */
gettype:
/* A label used as a jump-in point for a wrong response to the next question.*/
/* The procedure returns here when the entered table type isn't an M or I.  */
type "";
type "Is this chart for Individual or Multiple determinations? ";
table_type=gettext(" [Enter I or M please.]");
/*  These statements type out a message and request input of the form "I"   */
/*   or "M". The string input is assigned to variable 'table_type'. This    */
/*   variable is used as a flag for the need to expand the procedure for    */
/*   more data and an extra QC chart.                                       */
type "";
table_type=caps(table_type);
/* The function CAPS capitalizes a string, so I am using all caps here.     */
type "";
if table_type = "I" or table_type = "M" then
      goto check_group_table;
      else
        begin;
        type "I need either an I or an M.";
        type "";
        goto gettype;
        end;
/* This section checks that the table type entered is one of the two        */
/* acceptable types. If it is, the procedure goes on to the next line (from */
/* here). If not, it prints an error message and goes back to GETTYPE.      */
check_group_table:
/* The jump-in point from the 'standard' test section above.        */
gchart_name = '#'.chart_name;
/* Create the name of the grouphome table by attaching the #.   */
/* This uses a grouphome subdirectory called 'QC'.                  */
if tableexists(gchart_name) then
      begin;
      erase;
      type "";
      type "That QC table is already in the grouphome.";
      type "";
      type "Please check the name and try again.";
      type "";
      goto exiter;
      end;
/* The IF tests to see if the core table is present. If it is, the procedure */
```

```
/* prints an error message, and goes to the label EXITER                    */
/* (found at the end of the procedure) which stops the procedure.           */
call #make_temp_data( chart_name,table_type, 'M', std_flag, control_row);
/* The call to the interactive data entry subprocedure.                     */
/* the variable listed in the call have the standard meanings. The 'M' is a */
/* call flag telling the subprocedure which routine called it.              */
copy table('temp_data') to table(gchart_name);
/* Copies TEMP_DATA to the previously named final data table GCHART_NAME.   */

call #build_qc_charts( chart_name, table_type, 'M',std_flag,control_row,0);
/* Calls the chart building subroutine. Passes the data table name, the     */
/* table type, and the call-flag value (here='M').                          */
if yesanswer( "Do you want hardcopy? ") then call #qc_hardcopy(chart_name, 'M');
/* The user answers "Y" if hardcopy is desired.                             */
/* The call flag 'M' is again passed, since QC_HARDCOPY is used by other    */
/* procedures.                                                              */
del temp_data;
/* Clean up by deleting the temporary data table.                           */
exiter:type "All finished. See you next time.";
/* a final message to the user indicating the procedure is done.            */
```

PROCEDURE 3. #ADD_POINTS_TO_QC_CHART

```
/* Procedure ADD_POINTS_TO_QC_CHART                                        */
/* This is a procedure to modify Individuals or Mean/St. Dev. QC charts.   */
/*     Originator: Kirk L. Shanahan    Date: 8/14/89                       */
/*  Version 0   */
/*  Version 0.01  Fixed data correction bug.  9/8/89  KLS     */
/*                Added version printout      9/8/89  KLS     */
/*                Fixed up banner             9/8/89  KLS     */
/*  Version 1.0  Major revision - deleted grouphome subdirectories   */
/*                              - added user input of control limits */
/*                              - prettied up charts                 */
/*                              - added timestamp of all data        */
/*               reprogramming by Kirk Shanahan - start 9/19/89      */
/*              Major Revision - Began exclusive GROUPHOME utilization */
/*                              - split part of procedure off into subroutine*/
/*                                called BUILD_QC_CHARTS (BQC)              */
/*                              - added a call flag so BQC would work with  */
/*                                MQC and AQC.                              */
/*               reprogramming by Kirk Shanahan - start 9/25/89      */
/*                                                                         */
/*  Version 2.0  Major revision - Modified call to BUILD_QC_CHARTS to include */
/*                                DISPLAY_FLAG variable (set to 0), for     */
/*                                compatibility with MULTIPLE_COMPONENT_QC  */
/*                                _CHART procedure.                         */
/*                                Kirk Shanahan  3/1/90                     */
/*                                                                          */
/*  Version 2.1  Major Revision - Modified to use a sub-procedure           */
/*                                (QC_Y_AXIS_SETUP) to set Y axis low and   */
/*                                high. 7/6/90   KLS                        */
/*                                                                          */
/* This procedure modifies Shewhart-type Control charts, previously made via */
/* MAKE_QC_CHART, with either individual or multiple determination data. It  */
/* can use a list of 'standard' tests that are stored in a CONTROL_LIMITS    */
/* table, or it can modify a stand-alone chart for 'non-standard' tests. The */
/* CONTROL_LIMITS table has the following information: Name of test, Control */
/* Limit (3 sigma as a percent of the standard's value), Units of the test   */
/* (Y axis label), Table-type (individuals or multiples), a chart Title, and */
/* the number of points per line when the data is of the multiples type.     */
/*                                                                           */
/* The procedure prompts the user for the name of the test via a direct      */
/* question or via a listing of the standard tests, then prompts the user    */
/* for the appropriate data (done in subprocedure MAKE_TEMP_DATA). Next, it  */
/* copies the new data to the grouphome and calls the subprocedure (BUILD_   */
/* QC_CHARTS) which constructs the graphs and does standard Western Electric */
/* trend analysis, and finally asks the user if hardcopy output is desired   */
/* (achieved via subprocedure QC_HARDCOPY).                                  */
/*                                                                           */
/* The current implemetation of this procedure is via a menu system taken    */
/* from the QCA option to RS/1. The core chart building and trend analysis   */
/* procedures used in this procedure also came from the RS/1 QCA option.     */
/*                                                                           */
/*                                                                           */
/*  START OF PROCEDURE ADD_POINTS_TO_QC_CHART. CURRENT VERSION 2.1           */
/*                                                                           */
```

```
erase;
/*  Clear the screen  */
type "";
/* This construct types a blank line. It is used for spacing on the screen.  */
type "";
type "";
type "";
type "";
type "";
type "Welcome to the QC world. This is procedure ADD_POINTS_TO_QC_CHART (AQC).";
type "AQC will prompt you for new data and modify the old data table.";
type "AQC will then modify the appropriate QC chart(s).";
type "";
type "  Version 2.1";
type "";
type "";
/* Banner - contains procedure name so user can verify he is in the right */
/*          one.                                                          */
if tableexists('temp_data') then delete temp_data;
/*  temp_data is the temporary data table that this procedure uses for    */
/*  data entry. It eventually is renamed to a permanent table in the      */
/*  GROUPHOME.                                                            */
getname:
/* This is a label used as a jump-in point from a later GOTO stmt.        */
/*  It is the starting point of the procedure.                            */
IF YESANSWER('Are you adding points to a standard test?') THEN
/* If the user answers "Yes" here, the procedure goes to the CONTROL_LIMITS  */
/* table and displays a list of standard tests for the user to choose from.  */
/* A "No" sends the procedure to the section that gets a name directly from  */
/* the user.                                                                 */
 {
type"";
type "A list of methods will now be displayed.";
type "You should locate the test you want and remember its row number.";
type "You will be asked to enter the row number after the display is finished.";
type "If the test you want is not on the list, it is not a 'standard' test.";
type "Enter a 1 (choose the NON_STANDARD choice) to go back to the start.";
type "";
checker1: if not yesanswer("Are you ready?") then go to checker1;
/* The TYPE-CHECKER1 block above is meant to give the user a chance to get   */
/* for the upcoming display.                                                 */
/* The table in this statement is the CONTROL_LIMITS table. Here it is stored */
/* in a grouphome subdirectory and 'personalized' via the 'tnx' designation  */
/* for the DWPF Analytical Lab QC effort.                                    */
   get_test_name:
   DIS COL 1 OF #tnx_CONTROL_LIMITS;
/* Displays the list of test names.                                          */
   control_row= GETNUMBER('Please choose which test by entering the number:');
/* The user enters the row number of the test for which a chart is to be     */
/* constructed.                                                              */
      if control_row=1 then goto getname;
/* The CONTROL_LIMIT table should always have row 1 col 1 be "NON_STANDARD". */
/* This allows the user the chance to go back to the beginning when needed   */
/* via the GOTO above.                                                       */
   dis row control_row of col 1 of #tnx_control_limits;
   if not yesanswer('Is this the test you wanted?') then goto get_test_name;
```

```
/* Display the user's choice and confirm it.                          */
   chart_name = row control_row col 1 of #tmx_control_limits;
   table_type = caps(row control_row col 4 of #tmx_control_limits);
/* Get the stored info CHART_NAME and TABLE_TYPE on the test from the CONTROL */
/* _LIMITS table.                                                     */
   std_flag=TRUE;
/* STD_FLAG is a flag used in the subroutines to tell if the test is a  */
/* 'standard' test.                                                   */
   goto check_group_table;
/* Jump past the next block because it is for a 'non-standard' test.  */
 }
std_flag=FALSE;
control_row=0;
/* Since this section is reached only for 'non-standard' tests, the flag  */
/* indicating a 'standard' test is set FALSE, and the CONTROL_ROW variable */
/* is set to 0, so that it will not cause problems later if left empty.    */
chart_name =GETTEXT("What is the name of the QC Chart you wish to modify? ");
/* Chart_name is the variable that holds the core of the table and QC chart */
/* names. GETTEXT is an RPL funtion for terminal I/O that gets a string.   */
/* Here the input string is assigned to 'chart_name'.                      */
gettype:
/* A label used as a jump-in point for a wrong response to the next question.*/
/* The procedure returns here when the entered table type isn't an M or I.  */
type "";
type "Is this chart for Individual or Multiple determinations? ";
table_type=gettext(" [Enter I or M please.]");
/*  These statements type out a message and request input of the form "I"  */
/*   or "M". The string input is assigned to variable 'table_type'. This   */
/*   variable is used as a flag for the need to expand the procedure for   */
/*   more data and an extra QC chart.                                      */
type "";
table_type=caps(table_type);
/* The function CAPS capitalizes a string, so I am using all caps here.    */
type "";
if table_type = "I" or table_type = "M" then
      goto check_group_table;
      else
        begin;
        type "I need either an I or an M.";
        type "";
        goto gettype;
        end;
/* This section checks that the table type entered is one of the two       */
/* acceptable types. If it is, the procedure goes on to the next line (from */
/* here). If not, it prints an error message and goes back to GETTYPE.     */
check_group_table:
/* The jump-in point from the 'standard' test section above.        */
gchart_name = '#'.chart_name;
/* Create the name of the grouphome table by attaching the #.   */
/* This uses a grouphome subdirectory called 'QC'.                  */
if not tableexists(gchart_name) then
      begin;
      erase;
      type "";
      type "That QC table isn't in the grouphome.";
      type "";
```

```
      type "Please check the name and try again.";
      type "";
      goto exiter;
      end;
/* The IF tests to see if the core table is present. If it isn't, the      */
/* procedure prints an error message, and goes to the label EXITER         */
/* (found at the end of the procedure) which stops the procedure.          */
if table_type='I' AND lastcol(gchart_name)>11 then
      {
      erase;
      type "";
      type "The table type you supplied and the table itself don't match.";
      type "";
      type "Please check the name and/or table and try again.";
      type "";
      goto exiter;
      }
if table_type='M' and lastcol(gchart_name)=11 then
      {
      erase;
      type "";
      type "The table type you supplied and the table itself don't match.";
      type "";
      type "Please check the name and/or table and try again.";
      type "";
      goto exiter;
      }
/* The above section checks to make sure the user has not confused two tests. */
/* It checs the grouphome data table to verify that the entered and actual    */
/* table-type are the same. If they are, the procedure goes on, otherwise it  */
/* prints an error message and goes to the end of the procedure (label        */
/* EXITER).                                                                   */

call #make_temp_data( chart_name,table_type, 'A', std_flag, control_row);
/* The call to the interactive data entry subprocedure. The variables have   */
/* implied meaning. The 'A' is a call flag that lets the subprocedure know   */
/* which procedure called it.                                                */
add rows to table(gchart_name) from temp_data;
/* Adds the new rows to the old grouphome data table.                        */
call #build_qc_charts( chart_name, table_type, 'A',std_flag,control_row,0);
/* Calls the chart building subroutine. Passes the data table name, the      */
/* table type, and the call-flag value (here='A').                           */
if yesanswer( "Do you want hardcopy? ") then call #qc_hardcopy(chart_name,'A');
/* The user answers "Y" if hardcopy is desired.                              */
del temp_data;
/* Clean up by deleting the temporary data table.                            */
exiter:type "All finished. See you next time.";
/* A message letting the user know the procedure is done.                    */
```

PROCEDURE 4. #BUILD_QC_CHARTS

```
procedure(chart_name,table_type, call_flag,std_flag,std_row,display_flag);
/* This procedure builds Shewhart-type QC charts from a data table TEMP_DATA. */
/* The data in TEMP_DATA can be either add-on data to a pre-existing group-   */
/* home data table, or totally new data. The procedure searches TEMP_DATA for */
/* each unique region. A unique region is defined by a unique set of standard */
/* value and control limits. It then piecewise adds (creates if the first     */
/* time) regions to the QC chart. It sets X and Y axis parameters in an       */
/* attempt to optimize display, and creates messages and ticks containing     */
/* information about the regions. It then performs trend analysis.            */
/* Passed parameters:                                                         */
/*  chart_name -  actual data table in grouphome                              */
/*  table_type -  "I" or "M"                                                  */
/*  call_flag  - "A" or "M" depending on calling routine                      */
/*  std_flag - TRUE or FALSE, indicates if chart is for 'standard' test       */
/*  std_row - row in the CONTROL_LIMITS table holding the info on the         */
/*                 'standard' test                                            */
/*  display_flag - A flag that skips graphical display when it = -1           */
/*                                                                            */
/* Originator: Kirk L. Shanahan version 0. Date: c. 10/10/89                  */
/* Installed as BUILD_QC_CHARTS in ADS GROUPHOME 11/20/89                     */
/*                                                                            */
/* Modified to add DISPLAY_FLAG variable for multiple component standards     */
/* procedure - 3/1/90                          Version 1.0 now                */
/*                                                                            */
/* BEGINNING OF SUB-PROCEDURE BUILD_QC_CHARTS. VERSION 1.0                    */
/*                                                                            */
erase;
/* Clears the screen.                                                         */
gchart_name='#'.chart_name;
/* Constructs the name of the grouphome data table. Here a subdirectory is    */
/* used.                                                                      */
type "Working...";
/* A message indicating entry to this sub-procedure.                          */
table_type = caps(table_type);
if table_type='I' then
     {
     chart_type='IA';
     datacol=8;
     }
 else
     {
     chart_type='MA';
     datacol=1;
     }
/* Set up of some parameters used in the call to the QCA procedures           */
/* $MAKECHART and $ADDPOINTS.                                                 */
/* See the QCA manual for an explanation of the parameters.                   */
     qc_graph1 = gchart_name.'_qc_chart';
     qc_graph2 = gchart_name.'_qc_chart_stdev';
     if table_type='M' then qc_graph1 = qc_graph1.'_means';
/* Here I create two variables that have the core data table name as their    */
/* first letters, and QC_CHART as the next letters. QC_GRAPH1 will be used    */
/* for the name of the INDIVIDUALS QC chart. QC_GRAPH2 will be used for the   */
```

```
/* name of the STDEV QC chart (for 'Multiples' data).                    */
/* For 'Multiples' data, the letters "_MEANS" are added to identify the  */
/* table type in the display of a DIR command.                           */
if tableexists('temp_rows') then delete table temp_rows;
allocate table temp_rows 1 row by 1 col;
/* The table TEMP_ROWS is used to hold information required to build the */
/* the QC charts and put messages on them.                               */
a= row 1 col 4 of temp_data;
b= row 1 col 5 of temp_data;
/* A and B are used as checkpoints. Each row of the data table is checked */
/* for different control limit characteristics against A and B.          */
/* TEMP_ROWS will have 1 row for each region on the chart. A region is    */
/* defined by differing control limits. The first region is always the    */
/* started by the first row of the data table, thus the control limits and */
/* start point in the data table TEMP_DATA is stored.                    */
/* TARGET in col 1, RANGE in col 2, region starting row number in col 3.  */
     row 1 col 1 of temp_rows = a;
           /* Standard's value  */
     row 1 col 2 of temp_rows = b;
           /* Standard's range  */
     row 1 col 3 of temp_rows= 1;
          /*  message x position and new region start point */
     row 1 col 5 of temp_rows=
                'SID = '.a.' +/- '.b.'%';
          /* Message text  */
     row 1 col 6 of temp_rows=(a-3*b) -
                  1.1*chars(row 1 col 5 of temp_rows)/2;
          /* Message Y position */
temp_row=1;
/* TEMP_ROW is the row index of the table TEMP_ROWS.                     */
do i= 1 to lastrow('temp_data');
/* This DO loop will check every row of TEMP_DATA to find where changes in */
/* control limits occurs.                                                */
if ( (row i col 4 of temp_data = a ) AND (row i col 5 of temp_data = b) ) then
     donext;
/* If the standard's value and range are unchanged, go on to next row.   */
  else
     begin;
     temp_row = temp_row + 1;
     a= row i col 4 of temp_data;
     b= row i col 5 of temp_data;
     row temp_row col 1 of temp_rows = a;
           /* Standard's value  */
     row temp_row col 2 of temp_rows = b;
           /* Standard's range  */
     row temp_row col 3 of temp_rows= i;
          /*  message x position and new region start point */
     row (temp_row-1) col 4 of temp_rows = (i-1);
          /* Previous region's stop point  */
     row temp_row  col 5 of temp_rows=
               'SID = '.a.' +/- '.b.'%';
          /* Message text  */
     row temp_row col 6 of temp_rows=(a-3*b) -
                  1.1*chars(row 1 col 5 of temp_rows)/2;
          /* Message Y position */
/* This block is executed only when a change in standard's value or range is */
```

```
/* encountered. When this is found, first the row counter is incremented,    */
/* then the test variables A anb B are reset to the new values. Then the new */
/* test values are stored in the new row of TEMP_ROWS, and finally the stop  */
/* point of the old region is stored in col 4 of the previous row.           */
     end;
end;
/* End of DO loop.                                  */
row lastrow('temp_rows') col 4 of temp_rows = lastrow('temp_data');
/* The very last stop point entered into TEMP_ROWS will always be the last  */
/* row of TEMP_DATA, even if TEMP_ROWS only has 1 line.                      */
a= lastrow(gchart_name) - lastrow('temp_data');
/* If there was an old chart, a >0, otherwise a=0   */
if ( (row 1 col 1 of temp_rows = row a col 4 of table(gchart_name)) AND
    (row 1 col 2 of temp_rows = row a col 5 of table(gchart_name))  ) then
       row 1 col 5 of temp_rows = empty;
/* Row 0 is texts, will not equal a number.  */
/* Blank the message if not needed.          */
maker:
if chart_type= 'SA' then qcgraph= qc_graph2;
     else qcgraph=qc_graph1;
/* The following section of code is used twice for 'Multiples' data. Thus   */
/* the graph name is generalized to QCGRAPH for that purpose.               */
     ATTCKS = QCGRAPH.'@XTICKS';
     BMSGS  = QCGRAPH.'@MSGS';
     DATA_TABLE=QCGRAPH.'@DATA';
/* Construct some names of the graph's underlying tables.                 */
     if objSexists(data_table) then last_data= lastrow(data_table);
       else
         last_data =0;
/* A count of previous data is used below. It is obtained from the LASTROW  */
/* function on the graph's data table (if it exists).                       */
   if objSexists(qcgraph) then
     {
     start_row = 1;
     goto adder;
/* If the graph exists, no call to $MAKECHART is required, thus the procedure */
/* skips to ADDER.                                                           */
     }
    else
     {
     start_row= 2;
     c=1;
     d= row 1 col 4 of temp_rows;
     lot_id=rows c to d of col 7 of temp_data;
     v1= row 1 col 1 of temp_rows;
     v2 = v1*row 1 col 2 of temp_rows/300;
     v3 = row c to d of col datacol of temp_data;
     v4=EMPTY;
/* initially set up for 'IA' type                                        */
/* See the QCA manual for an explanation of the parameters.              */
     if chart_type= "MA" then
/* Change the setup for a 'Multiples' type chart.                        */
/* See the QCA manual for an explanation of the parameters.              */
     {
     v2 = row 1 col 3 of temp_data;
     v4 = v1*row 1 col 2 of temp_rows/300;
```

```
      }
      if chart_type= "SA" then
/* Change the setup for a 'Multiples' type standard deviation chart.    */
/* See the QCA manual for an explanation of the parameters.             */
      {
      v1 = row 1 col 3 of temp_data;
      v2 = row c to d of col datacol of temp_data;
      v3 = row 1 col 1 of temp_rows*row 1 col 2 of temp_rows/300;
      v4 = EMPTY;
      }
 call $makechart( qcgraph, lot_id, chart_type, v1, v2, v3, v4, EMPTY, FALSE);
/* The noninteractive call to the RS/1 QCA package procedure $MAKECHART.    */
/* See the QCA manual for an explanation of the parameters.                 */
      call obj$setvalue(qcgraph,"dismods",'box nokey');
/* Sets up the graph's display modifiers permanently.                       */
if d=1 then row 2 of table(data_table)=row 1 of table(data_table);
/* BUGNOTE!!! D=1 implies only 1 point is put in the chart by $MAKECHART,   */
/* however, $ADDPOINTS assumes at least two are to be present. $MAKECHART   */
/* adds a second row to the chart's data table. Unfortunately, $MAKECHART   */
/* doesn't completely fill that second row and $ADDPOINTS needs it filled.  */
/* Therefore I do it for $MAKECHART.                                        */
      if lastrow('temp_data') <= 5 then xpos=2; else xpos=4;
/* XPOS will be the X position of the curve labels.                         */
/* We are still in the $MAKECHART section here, and we are preparing to     */
/* add messages and ticks to the graph.                                     */
      xpos2=empty;
      do i = 1 to lastrow('temp_rows');
        if row i col 4 of temp_rows >= xpos then
                {
                xpos2= i;
                doexit;
                }
      end;
      if xpos2=empty then xpos2 = 1;
/* This loop finds the row in TEMP_ROWS, representing a region on the QC    */
/* chart, that contains the 'default' positions for the control line        */
/* messages. This row is then used to set the Y positions of the messages   */
/* in the following lines of code.                                          */
      row 1 col 1 of table(bmsgs)='UPPER LIMIT';
/* Enter the first message into the message table.                          */
      if chart_type <> 'SA' then
       row 1 col 3 of table(bmsgs)=
        row xpos2 col 1 of temp_rows*(1 + row xpos2 col 2 of temp_rows/100) -
        .1 * ( row xpos2 col 1 of temp_rows* row xpos2 col 2 of temp_rows/100 );
          else
        row 1 col 3 of table(bmsgs)=
         1.7*row xpos2 col 1 of temp_rows*( row xpos2 col 2 of temp_rows/300);
/* Calculate the first message's Y position, based on table type.           */
/* The '1.7' in the ELSE part of the statement comes from the way QCA draws */
/* control lines on the "SA" type charts. Found in a QCA manual appendix.   */
/* There is a little offset from the manual value, so the message is not    */
/* written over the line itself. The other part of the IF uses a direct     */
/* calculation of the control line position and lowers the message by 10% of */
/* the difference between the mean and upper limit lines.                   */
      row 2 col 1 of table(bmsgs)='STD VALUE';
/* Enter the second message into the message table.                         */
```

```
      if chart_type ◇ 'SA' then
        row 2 col 3 of table(bmsgs)= row xpos2 col 1 of temp_rows -
        .1*(row xpos2 col 1 of temp_rows * row xpos2 col 2 of temp_rows/100 );
       else
         row 2 col 3 of table(bmsgs)=
          .85*row xpos2 col 1 of temp_rows*( row xpos2 col 2 of temp_rows/300);
/* Again, the Y position of the second message is calculated, with offsets.  */

      row 3 col 1 of table(bmsgs)='LOWER LIMIT';
/* Enter the third message into the message table.                          */
      if chart_type ◇ 'SA' then
        row 3 col 3 of table(bmsgs)=
         row xpos2 col 1 of temp_rows*(1 - row xpos2 col 2 of temp_rows/100) -
         .1* ( row xpos2 col 1 of temp_rows*row xpos2 col 2 of temp_rows/100 );
       else
        row 3 col 3 of table(bmsgs)=.1;
      }
/* Again, the Y position of the third message is calculated, with offsets.  */
        row 1 to 3 of col 2 of table(bmsgs)= xpos;
/* The X position of all three messages is set with this statement.         */
/* This marks the end of the $MAKECHART section.                            */
adder:
/* The following section is where new regions are added to pre-existing QC  */
/* charts (which may have just been created above).                          */
do i= start_row to lastrow('temp_rows');
/* START_ROW was set above. It is the row in TEMP_ROWS where addition of    */
/* other regions will start. If the data being added is new, START_ROW = 2, */
/* since the first row is used by $MAKECHART above. Otherwise START_ROW=1.  */
      c= row i col 3 of temp_rows;
      d= row i col 4 of temp_rows;
/* C and D are the starting and stopping data point for the current region. */
/* They are the row numbers in TEMP_DATA.                                   */
      lot_id= row c to d of col 7 of temp_data;
/* A tableportion stored in a variable, for use as a parameter in $ADDPOINTS. */
/* See the QCA manual for $ADDPOINTS parameter definitions.                 */

      if chart_type = 'IA' then
        {
        datacol=8;
        a1 = row c to d of col datacol of temp_data;
        a2 = EMPTY;
        v1 = row i col 1 of temp_rows;
        v2 = row i col 2 of temp_rows*v1/300;
        v3 = EMPTY;
        }
      if chart_type='MA' then
        {
        a1 = row c to d of col datacol of temp_data;
        a2 = rows c to d of col 2 of temp_data;
        v1 = row i col 1 of temp_rows;
        v2 = v1*row i col 2 of temp_rows/300;
        v3 = row 1 col 3 of temp_data;
        }
      if chart_type= 'SA' then
        {
        a1 = row c to d of col 2 of temp_data;
```

```
        a2 = row c col 2 of table temp_data;
        v1 = row i col 1 of temp_rows*row i col 2 of temp_rows/300;
        v2 = row 1 col 3 of temp_data;
        v3 = EMPTY;
        }
/* Based on the chart-type, the above code sets up the $ADDPOINTS parameters */
/* with the appropriate values. See the QCA manual for explanations.        */
 call $addpoints( qcgraph, "NEW", lot_id, a1, a2, EMPTY, v1, v2, v3, False);
/* The non-interactive call to the $ADDPOINTS procedure. See the QCA manual */
/* for a description of the parameters.                                     */
end;
/* End of the DO loop that adds regions to the QC chart.                    */
     allocate table table(aticks) 1 row by 1 col ;
/* Set up the XTICKS table.                                                 */
     set units of x axis of graph(qcgraph) to 1;
/* Set the units on the X axis to whole numbers only.                       */
     if lastrow(gchart_name)> 30 then
          {
          low of x axis of graph(qcgraph) = lastrow(gchart_name)- 30.25;
          rows 1 to 3 of col 2 of table(bmsgs) = lastrow(gchart_name)- 26.25;
          do i = (lastrow(gchart_name)-30) to 1 by -1;
            if row i col 3 of table(data_table) ◇ empty then doexit;
          end;
/* The IF statement limits the displayed part of the QC chart to the last 30 */
/* points. It also moves the limit messages to 4 units in from the left.    */
/* The DO loop inside the IF finds the first non-zero entry in the control  */
/* limits column. When it finds a non-zero number, it exits, leaving the    */
/* loop index I at that value. I is used in the next line.                  */
          delta = (row i col 3 of table(data_table)- row i col 4 of
                    table(data_table))*.1;
/* DELTA is used below to reposition the control limit line messages in Y.  */
     row 1 col 3 of table(bmsgs) = row i col 3 of table(data_table)-delta;
     row 2 col 3 of table(bmsgs) = row i col 4 of table(data_table)-delta;
     row 3 col 3 of table(bmsgs) = row i col 5 of table(data_table)-delta;
/* Based on the value of DELTA, the Y positions of the messages is          */
/* recalculated.                                                            */
          }
       else
          low of x axis of graph(qcgraph) = .75;
/* If there are less than 30 points, the Y axis low is set to .75.          */
     LABEL OF X AXIS OF GRAPH(qcgraph)="DATE";
/* Sets the label of the X axis.                                            */
     COLOR OF CURVES 2 OF GRAPH(qcgraph)="BLUE";
     COLOR OF CURVES 4 OF GRAPH(qcgraph)="BLUE";
/* Sets up some default colors for the control limit lines on the chart.    */
     xhigh = (lastrow(gchart_name)+.6);
/* Set up the XHIGH variable to give enough room to display any STANDARD    */
/* messages.                                                                */
     high of x axis of graph(qcgraph) = xhigh;
/* Set the X axis high.                                                     */
     xlow = integer(low of x axis of graph(qcgraph))+1;
/* The X axis LOW is set above to a fraction less than either 1 or some other */
/* value. The INTEGER function truncates the value, and then 1 is added to  */
/* move the value back up to the first displayed point.                     */

     xhigh = integer(xhigh);
```

```
/* INTEGERize the XHIGH value to get rid of the .6 added above.          */
   call #qc_y_axis_setup(qcgraph, xlow, xhigh);
/* Call the sub-procedure that sets the Y axis LOW and HIGH values.      */
   yhigh= high of y axis of graph(qcgraph);
   ylow = low of y axis of graph(qcgraph);
/* Get the new low and high values and save for later use.               */
   last_data = lastrow(gchart_name)-lastrow('temp_data');
/* Calculate the old last row of data for later use.                     */
   c= lastrow(aticks);
/* Get the current last row of the tick table in preparation for adding 'NEW */
/* STD' markers on the X axis.                                           */
   if c ◇ 0 then
/* If C◇0, a previous tick table exists. Therefore the procedure needs to    */
/* check the last old standard against the new first standard. If they are   */
/* different, a new marker must be added.                                */
    if ((row 1 col 1 of temp_rows ◇ row last_data col 4 of table(gchart_name))
     OR (row 1 col 2 of temp_rows ◇ row last_data col 5 of table(gchart_name)))
/* Testing for non-equivalence in the old last and new first standards.  */
    then
      {
      c=c+1;
      row c col 1 of table(aticks)= 'NEW STD';
      row c col 2 of table(aticks)= .5 + last_data;
/* Here is where the old last row of data is used in positioning a marker. */
      row c col 3 of table(aticks)= 'GRID';
      row c col 4 of table(aticks)= 'GREEN';
      row c COL 5 OF TABLE(ATICKS)="VERTICAL";
/* Add a new marker by incrementing C and filling in the new row with the   */
/* correct information.                                                  */
      }
    do i= 1 to lastrow('temp_rows')-1;
/* The index on this DO loop is tricky. The actions of the loop drive off of */
/* C, which is incremented immediately, thus I only controls the incrementing.*/
      c=c+1;
      row c col 1 of table(aticks)= 'NEW STD';
      row c col 2 of table(aticks)= row i col 4 of temp_rows + .5 + last_data;
      row c col 3 of table(aticks)= 'GRID';
      row c col 4 of table(aticks)= 'GREEN';
      row c COL 5 OF TABLE(ATICKS)="VERTICAL";
/* Add a new marker for each control limit region (each row in TEMP_ROWS).  */
    end;
    do i= 1 to lastrow('temp_data');
      c=c+1;
      row c col 1 of table(aticks)= row i col 7 of temp_data;
/* Set the X tick labels equal to the dates from TEMP_DATA.              */
      row c col 2 of table(aticks)= last_data + i;
/* Set the X poistion of the ticks to the subgroup number.               */
      row c COL 5 OF TABLE(ATICKS)="VERTICAL";
/* Set the display attribute of the X tick.                              */
    end;
      lr_msgs=lastrow(bmsgs);
      do i= 1 to lastrow('temp_rows');
      row (i+lr_msgs) col 1 of table(bmsgs)= row i col 5 of temp_rows;
      row (i+lr_msgs) col 2 of table(bmsgs)=
           row i col 3 of temp_rows+.25 + last_data;
      row (i+lr_msgs) COL 9 OF TABLE(bmsgs)= "VERTICAL";
```

```
      row (i+lr_msgs) COL 10 OF TABLE(bmsgs)= "LEFT";
      end;
/* The above DO loop adds the 'STD=' messages to the chart by adding rows to */
/* the MSGS table of the QC graph.                                            */

      yheight = 1.07 * (yhigh - ylow) /8.25;
/* 8.25 is the height (in inches) of the graph on paper, 1.07 = 1/2 of the   */
/* height of the 'STD' message (in inches).                                   */
      do i= 4 to lastrow(bmsgs);
      row i COL 3 OF TABLE(bmsgs)= ylow + 1.1*yheight ;
      end;
/* The above DO loop adjusts the centers of the 'STD=' messages according to */
/* the Y axis low. This DO loop allows the Y axis low to be set above without */
/* worrying about any new STD message.                                        */
      delete table(bmsgs) where col 1 = empty;
/* Cleans up the MSGS table, if required(?).                                  */

/* The next section does the TREND analysis of the QC chart. Full Western    */
/* Electric rules for MEANS and INDIVIDUALS. Restricted set for SIDEVS.       */
      c=lastrow(data_table);
/* Get the end of the data to mark the region to TREND.                       */
      GRAPHNOTES of graph(qcgraph) = empty;
/* Empty out any previous results.                                            */
      if chart_type <> 'SA' then
/* For MEANS and INDIVIDUALS.                                                 */
        {
        if call_flag='A' then delete curve 5 of graph(qcgraph);
/* If this is not the first time, delete the old trending info.               */
        call $trend( qcgraph, 1,  c, false);
/* The non-interactive call to $TREND. This starts at subgroup 1 and goes to */
/* subgroup C. One could start anywhere, but beware that there is a bug.      */
/* BUGNOTE!!!  $TREND  (possibly $CALLRULES also)                             */
/* $MAKECHART and $ADDPOINTS do not totally fill in the column of data in the */
/* DATA table of the QC chart. They just fill in the start and stop points of */
/* a given region, leaving the lines in between empty. If the trend region    */
/* starts with one of these empty points, $TREND crashes. A fix would be to   */
/* add some code in your procedure to fill in the empty cells in the DATA     */
/* table of the graph. This may also occur with the $CALLRULES procedure.     */
        }
      if chart_type = 'SA' then
        {
        if call_flag='A' then delete curve 5 of graph(qcgraph);
        call $callrules( '#restricted_trend_rules', qcgraph, 1,  c, false);
/* $CALLRULES is the underlying procedure to $TREND. In $TREND a special      */
/* rules table is used. Here, I bypass $TREND to apply a different set of     */
/* rules, the rules table is #restricted_trend_rules in the grouphome          */
/* subdirectory. This trend analysis only marks points outside the control    */
/* limits.                                                                    */
        }
      if c>30 then
      GRAPHNOTES of graph(qcgraph) = "Current screen display starts at subgroup"
              .(c-30).GRAPHNOTES of graph(qcgraph) ;
/* Add a note to the graph note telling if display is windowed and where.     */
      crlf=num_to_ascii(13).num_to_ascii(10);
/* CRLF holds the non-printable characters <CR><LF>. It is used below for     */
/* spacing in text messages and notes.                                        */
```

```
      label of curve 1 of graph(qcgraph) = 'Data Points'.crlf;
      label of curve 2 of graph(qcgraph) = 'Upper Limit'.crlf;
      label of curve 3 of graph(qcgraph) = 'Standard'.crlf.'Value'.crlf;
      label of curve 4 of graph(qcgraph) = 'Lower Limit'.crlf;
      label of curve 5 of graph(qcgraph) = 'Out-of-Control'.crlf.'Point'.crlf;
/* Set up the spacing and labelling of the graph key.                    */
      if (std_flag AND chart_type <> 'SA') then graphtitle of graph(qcgraph)=
         row std_row col 5 of #tmx_control_limits;
      if (std_flag AND chart_type = 'SA') then graphtitle of graph(qcgraph)=
        'Standard Deviation Control Chart to go with'.crlf.
         row std_row col 5 of #tmx_control_limits;
/* Sets the graph titles, according to chart type.                       */
      label of y axis of graph(qcgraph) =
          row std_row col 3 of #tmx_control_limits;
      orientation of y axis of graph(qcgraph) = 'VERTICAL';
/* Set display characteristics of the Y axis.                            */
if display_flag = -1 then goto skipped_display;
/* This line skips the graphics diplay if DISPLAY_FLAG is set at -1.     */
c=ext ( 1, 5, caps ( global('TERMINAL') ) );
if c = 'IBMPC' then call #ibmpc_fixup(aticks,qcgraph);
      else dis graph( qcgraph );
/* Get the RS/1 system variable TERMINAL and see if it starts with "IBMPC". */
/* (This uses the GLOBAL function to get it, the CAPS function to capitalize */
/* it, and the EXT function to get the first 5 characters.)              */
/* If it does, the user is using the RSTERM terminal emulator, which has a  */
/* problem with displays.                                                */
/* The subprocedure IBMPC_FIXUP temporarily alters the display and displays  */
/* the graph on the screen. Otherwise a simple DISPLAY suffices.         */
checker1: if not yesanswer("Continue? ") then goto checker1;
/* The checkpoint for display. Display is non-optional.                  */
skipped_display:
if chart_type = 'MA' then
/* If the chart type is MA, this was the first pass through on a 'Multiples' */
/* type data set. A STDEV chart needs to be constructed also. The chart type */
/* and  data column number are reset to the appropriate values, and the      */
/* procedure is sent back to the start to do it all over again on the        */
/* standard deviation data.                                              */
      begin;
      chart_type = 'SA';
      datacol = 2;
      goto maker;
      end;
delete table temp_rows;
/* Cleans up after itself by deleteing the temporary table TEMP_ROWS.       */
end;
```

PROCEDURE 5. #MAKE_TEMP_DATA

```
procedure(chart_name, table_type, call_flag, std_flag, std_row);
/* This subprocedure interactively prompts the user for the necessary      */
/* information to construct a temporary data table, TEMP_DATA, in the      */
/* format used in the QC procedures MAKE_QC_CHART and ADD_POINTS_TO_QC_    */
/* CHART. The passed parameters are : the core test name (CHART_NAME),     */
/* the table_type (I or M), a call-flag (A or M), a 'standard' test flag   */
/* (STD_FLAG, TRUE or FALSE), and a row number in the CONTROL_LIMITS table */
/* where 'standard' test information is stored.                            */
/*                                                                         */
/* TEMP_DATA is eventually copied to or added to the grouphome data table. */
/*                                                                         */
/* Version 0. - originally an integral part of version 0 of MAKE_QC_CHART  */
/* and ADD_POINTS_TO_QC_CHART.                                             */
/*                                                                         */
/* BEGINNING OF SUB-PROCEDURE MAKE_TEMP_DATA. VERSION 0.                   */
new_flag=TRUE;
/* A logical flag used to tell if the procedure should get a first value   */
/* for standards.                                                          */
gchart_name='#'.chart_name;
/* Builds the grouphome data table name.                                   */
colno=8;
if table_type="M" then
      {
      if std_flag then colno = row std_row col 6 of #trx_control_limits+7;
       else
        colno=getnumber("How many points per line? ")+7;
      }
/* COLNO is the variable that holds the maximum data table column number.  */
/* If the table is for "Multiples" data, i.e. more than 1 determination per */
/* sample, the number of points per line is obtained via the GETNUMBER     */
/* terminal I/O function for a 'non-standard' test, or from the CONTROL_   */
/* LIMITS table for a 'standard' test, and COLNO is adjusted appropriately. */
/*   NOTE:: COLNO is NOT the last column of the table!                     */
crlf=num_to_ascii(13).num_to_ascii(10);
/* CRLF is a text variable that holds the non-printable characters <CR><LF>. */
/* It is used in spacing the TEMP_DATA col 5 header below.                 */
allocate table temp_data 1 row by colno col;
row 0 col 0 of temp_data=chart_name;
row 0 col 1 of temp_data='MEAN';
row 0 col 2 of temp_data='ST DEV';
row 0 col 3 of temp_data='COUNT';
row 0 col 4 of temp_data='STDVALUE';
row 0 col 5 of temp_data='RANGE'.crlf.'% of STD';
row 0 col 6 of temp_data='STD REF';
row 0 col 7 of temp_data='DATE';
row 0 col 8 of temp_data='VALUE1';
/* Allocates space for the temporary data table TEMP_DATA and fills in     */
/* column headers.                                                         */
if colno>= 9 then
     {
     do j= 9 to colno;
     row 0 col j of temp_data='VALUE'.(j-7);
/* This DO loop enters the rest of the column headers for 'Multiples' data. */
```

```
/* The header is the word 'VALUE' with the value count added on the end     */
/* (i.e. VALUE2). It is only used if COLNO > 8, which should happen only if */
/* 'Multiples' data is being entered.                                        */
      end;
      }
watch cols 4 to colno of temp_data;
/* The RS/1 WATCH feature allows all changes made to the TEMP_DATA table to */
/* be displayed immediately.                                                 */
type "";
type "";
type "O.K., we are ready to go.";
type "";
type "";
/* A banner indicating everything has worked so far.                         */
temp_row=0;
/* A variable that is used to keep track of the number of rows entered.      */
enter_data:
/* This label indicates the start of the data entry section. It is the       */
/* entry point for a later GOTO, where the user has indicated more data is   */
/* to be entered than was originally estimated.                              */
row_number = getnumber("How many rows of data you will enter today? ");
/* ROW_NUMBER is used as the index of a DO loop (following) that controls    */
/* data entry into TEMP_DATA.                                                */
if row_number < 1 then
      {
      type"";
      type "Sorry I need a number greater than or equal to 1.";
      goto enter_data;
      }
/* If the operater enters less than 1 here, I presume a mistake has been     */
/* made. I tell the operator so, and go back to the data entry point.        */
get_first_standards:
if (call_flag = 'M' AND new_flag=TRUE )then
/* The subsequent section of code is only executed for a totally new chart. */
/* Thus the call_flag must be 'M' and the new_flag must be TRUE.             */
      {
      stdvalue=getnumber('What is the standard value for the first point? ');
      if std_flag then sd= row std_row col 2 of #trx_control_limits;
       else sd=getnumber(
        'What is the size of the control limit as a percent of the standard? ');
/* These two variables will hold the value for the standard and its range.   */
/* For a 'standard' test the standard deviation is derived from the CONTROL_ */
/* LIMITS table entry.                                                       */
      stdref=
      gettext('Please enter a notebook/pagenumber reference for the standard:');
      new_flag=FALSE;
/* Set new_flag so that this section is not executed again.                  */
      goto set_values;
      }
      if (call_flag='A' and new_flag=TRUE ) then
/* The subsequent section of code is only executed for an old chart on the   */
/* first pass only. Thus the call_flag must be 'A' and the new_flag must be  */
/* TRUE.                                                                     */
       {
        stdvalue=row lastrow(gchart_name) col 4 of table(gchart_name);
        sd = row lastrow(gchart_name) col 5 of table(gchart_name);
```

```
       stdref= row lastrow(gchart_name) col 6 of table(gchart_name);
/* If the call was from ADD_POINTS... and this is the first pass, the        */
/* standard's info is derived from the last row of the grouphome data table. */
       new_flag=FALSE;
/* Set new_flag so that this section is not executed again.                  */
       goto set_values;
      }

stdvalue=row lastrow('temp_data') col 4 of table('temp_data');
sd = row lastrow('temp_data') col 5 of table('temp_data');
stdref= row lastrow('temp_data') col 6 of table('temp_data');
/* On all passes through this section, except the very first, the standard's */
/* information is initially derived from the previous line in TEMP_DATA.     */
set_values:
do ii= 1 to row_number;
/* The DO loop that controls data entry.                                     */
temp_row=temp_row+1;
/*  Increment the total number of rows entered.                              */
row temp_row col 4 of temp_data = stdvalue;
row temp_row col 5 of temp_data= sd;
row temp_row col 6 of temp_data=stdref;
row temp_row col 7 of temp_data =
 getdate("Please enter a date for this determination (example 1/1 or 3/10/89): ");
/* A DATE must always be entered. It is stored in column 7 of TEMP_DATA.     */
a= getnumber("Please enter the determination value: ");
/* A NUMBER must always be entered. It is stored in column 8 of TEMP_DATA.   */
/* However, the value can also indicate an end to data entry. If it equals   */
/* -999, the following test deletes the current row, adjusts the total rows  */
/* entered count, prints a message, and goes to the label MOREENTER, where   */
/* the procedure asks if more data is to be entered.                         */
if a<>-999 then row temp_row col 8 of temp_data =a;
  else
      begin;
      del row temp_row of temp_data;
      temp_row= temp_row -1;
      type "";
      type "Exiting data entry section early.";
      type "";
      goto moreenter;
      end;
/* The IF test on the determination value entered. TRUE results in the       */
/* datum being stored in column 8, FALSE means delete the row, etc.          */
/*                                                                           */
/* This IF statement is an attempt to handle the problem of a user over-     */
/* estimating the amount of data to be entered.                              */
if colno <= 8 then goto get_standards;
/* This IF skips the next couple of lines if 'Individuals' type data is      */
/* being entered (based on a test of the maximum column number).             */
      do j= 9 to colno;
/* This DO loop enters the rest of the determinations for 'Multiples' data.  */
    row temp_row col j of temp_data = getnumber("Please enter the determination
value: ");
/* This statement gets the next number and writes it into TEMP_DATA.         */
      end;
/* End of the 'Multiples' data entry DO loop.                                */
get_standards:
```

```
if yesanswer('Is the current standard value OK? ',TRUE) then goto next_row;
/* This IF statement allows the user to change the standard's values if    */
/* desired. It presents a default answer of 'Yes', indicating no change, as */
/* this is the anticipated usual answer.                                    */
  else
    {
    row temp_row col 4 of temp_data =
      getnumber('What is the standard value for this point? ');
/* Get the new standard's value.                                            */
    if not std_flag then
      sd=getnumber('What is the control limit as a percent of the standard? ');
    row temp_row col 5 of temp_data= sd;
/* Get the new control limit, either from the user or from the CONTROL_LIMITS */
/* table.                                                                   */
    row temp_row col 6 of temp_data =
      gettext('What is the notebook/pagenumber reference for the standard? ');
/* Get the reference to where the new standard is documented.               */
      stdvalue  = row temp_row col 4 of temp_data;
      stdref = row temp_row col 6 of temp_data;
/* Saves the new values for the next line.                                  */
    }
next_row: type "";
/* Print a blank line before the next DATE/VALUE(S) data entry block.       */
end;
/* End of the data entry block. When the loop counter reaches the estimated */
/* count enterd above, the procedure goes on to the next statement below.   */
moreenter:
/* MOREENTER label, used as an entry point for the GOTO above that skipped  */
/* the 'Multiples' data entry.                                              */
if yesanswer("Do you need to enter more rows? ") then goto enter_data;
/* This IF is for when the user underestimates the number of data rows to   */
/* be entered. If more data is to be entered the procedure goes back to     */
/* the ENTER_DATA label above.                                              */
datacheck:
/* The DATACHECK label indicates the start of the data corection section of */
/* this procedure. The procedure gives the user the opportunity to change   */
/* any data that is wrong.                                                  */
dis col 4 to lastcol('temp_data') of temp_data;
/* Display all the data typed in this session.                              */
/* The user should look at this display to locate incorrect data.           */
if yesanswer("Does this all look correct? ") then
     goto ender;
/* If the data is all correct, the user answers "Y", and the procedure
   skips to the label ENDER.                                                */
    else
/* If the data is not all correct the user enters "N" above and the         */
/* procedure goes to the following section, which is marked with the label  */
/* FIXNUMBER.                                                               */
     begin;
     fixnumber:
/* The procedure requires the user to enter the bad datum's row and column  */
/* number, assigned to 'a' and 'b', respectively.                           */
     a=getnumber('What is the row number of the bad data?');
     IF A> LASTROW('TEMP_DATA') THEN
       {
       TYPE "";
```

```
        TYPE "The row number you entered was to large. Please try again.";
        checker1: if not yesanswer('Are you ready? ',TRUE) then goto checker1;
        goto datacheck;
        }
/* If the row entered exceeds the existing number of rows, the user has made */
/* a mistake and should start over.                                          */
      b=getnumber('What is the column number of the bad data?');
      IF b> lastcol('temp_data') THEN
        {
        TYPE "";
        TYPE "The column number you entered was to large. Please try again.";
        checker2: if not yesanswer('Are you ready? ',TRUE) then goto checker2;
        goto datacheck;
        }
/* If the column entered exceeds the existing number of columns, the user    */
/* has made a mistake and should start over.                                 */
  if b= 7 then
      row a col b of temp_data = getdate('What should I put there instead? ');
  else
   if b = 6 then
      row a col b of temp_data = gettext('What should I put there instead?');
    else
     if (std_flag and b=5) then
        {
        type 'Sorry, Changing the Limit is not allowed.';
        goto datacheck;
        }
      else
       row a col b of temp_data=getnumber('What should I put there instead? ');
/* If the datum's column number is 7, the datum is a DATE and the procedure  */
/* looks for a DATE specifically, otherwise it checks to see if the data     */
/* column is for the std. ref. information. If so, it gets a text string,    */
/* otherwise it gets a NUMBER to replace the bad datum. (However, it does    */
/* first check to see if the user is trying to change a standardized control */
/* limit. If he is, the procedure types an error message and goes back to    */
/* redisplay of the data typed in.) It then replaces the bad datum with the  */
/* new number/text/date.                                                     */
      display col (b-1) to (b+1) of rows (a-1) to (a+1) of temp_data;
/* The procedure attempts to display a 3x3 section of the data table that is */
/* centered on the recently corrected datum.                                 */
      if yesanswer("Is this OK now? ") then goto datacheck;
      goto fixnumber;
/* If the new datum is correct, the procedure goes back to DATACHECK and     */
/* to redisplay the data table (TEMP_DATA) for final confirmation.           */
/* Otherwise it skips back up to FIXNUMBER and asks for the row, column, and */
/* correct value again.                                                      */
/* THIS COULD BE USED AS A SHORT-CUT IN DATA CORRECTION IF THE USER WANTS    */
/* TO SKIP THE REDISPLAY.                                                    */
      end;
/* End of the logical block of the IF statement.                             */
ender:
/* The label ENDER marks the end of the data entry phase of the procedure.   */
/* After here, the procedure begins doing the chart building, and user input */
/* is used only to get optional printouts and graphics displays.             */
if lastcol('temp_data')>8 then
/* This IF statement is used only for 'Multiples' data. The DO loop          */
```

```
/* immediately below calculates the mean, standard deviation, and count for  */
/* each data table row.                                                      */
     do i= 1 to lastrow('temp_data');
/* DO for all of TEMP_DATA.                                                  */
     if count of row i of cols 8 to lastcol('temp_data') of temp_data >1 then
/* On the off chance the user has circumvented this procedure and            */
/* accidentally deleted some data, this IF checks to make sure a standard    */
/* deviation can be calculated (requires at least two data points.           */
    row i col 1 of temp_data=
     mean  of row i of cols 8 to lastcol('temp_data') of temp_data;
/* MEAN is stored in column 1.                                               */
       row i col 2 of temp_data =
         stdev of row i of cols 8 to lastcol('temp_data') of temp_data;
/* STDEV is stored in column 2.                                              */
       row i col 3 of temp_data =
         count of row i of cols 8 to lastcol('temp_data') of temp_data;
/* COUNT is stored in column 3.                                              */
     end;
/* End of DO loop.                                                           */
oper = strip($getjobinfo("username"));
/* This is a call to VMS for the username. The STRIP function removes        */
/* trailing blanks.                                                          */
colno=colno+3;
/* Increase the last column number in preparation for time and operator      */
/* stamping.                                                                 */
do i= 1 to lastrow("temp_data");
row i col (colno-2) of temp_data=date();
row i col (colno-1) of temp_data=time();
row i col colno of temp_data=oper;
end;
/* The DO loop time, date, and operator stamps the data table. */
row 0 col (colno-2) of temp_data = 'ENTDATE';
row 0 col (colno-1) of temp_data = 'ENTTIME';
row 0 col  colno    of temp_data = 'OPERATOR';
/* These statements label the data table columns                 */

end;
```

PROCEDURE 6. #IBMPC_FIXUP

```
procedure(aticks,qcgraph);
/* This procedure temporarily suppresses the DATE display on the X axis of  */
/* a QC chart. This is required when using the free BBN terminal emulator   */
/* RSTERM. RSTERM prints the dates at standard height, leaving no room for  */
/* the chart to be displayed. Therfore, the dates are temporarily replaced  */
/* subgroup numbers, the graph displayed, and the dates replaced.           */
/* This procedure should only be accessed when the TERMINAL variable begins */
/* with the letters 'IBMPC'.                                                */
/*                                                                          */
/* Version 0. Originator: Kirk L. Shanahan  Date: c. 9/20/89                */
/* Installed in ADS GROUPHOME as IBMPC_FIXUP on 11/20/89.                   */
/*                                                                          */
/* BEGINNING OF SUB-PROCEDURE IBMPC_FIXUP. VERSION 0.                       */
/*                                                                          */
if tableexists('temp_store') then delete table('temp_store');
make table temp_store from col 1 of table(aticks);
/* TEMP_STORE will temporarily hold the tick labels (col 1 of tick table).  */

do i= 1 to lastrow(aticks);
if row i col 1 of table(aticks)='NEW STD' then
     row i col 1 of table(aticks)='N';
  else row i col 1 of table(aticks)=row i col 2 of table(aticks);
end;
/* This DO loop sets the tick label to a number unless is is one of the     */
/* special ticks "NEW STD", in which case it abbreviates it to 'N'.         */

dis graph( qcgraph );
/* A non-optional display. The user should always see the trended QC chart.  */

type "DATE DISPLAY SUPPRESSED. DATES WILL SHOW IN HARDCOPY.";
/* This types a message to the user that the date display has been suppressed.*/

col 1 of table(aticks)=col 1 of temp_store;
/* Restores the old tick labels.                                            */
delete table temp_store;
/* Cleans up the TEMP_STORE table.                                          */
end;
```

PROCEDURE 7. #QC_HARDCOPY

```
procedure(chart_name, call_flag);
/* This procedure generates hardcopy output of QC charts, summary QC charts, */
/* or QA report charts. The passed variable CALL_FLAG distinguishes this.    */
/*                                                                           */
/* Originator: Kirk L. Shanahan  Date: c. 9/20/89  VERSION 0.                */
/* Installed in TNX GROUPHOME as QC_HARDCOPY  11/20/89                       */
/*                                                                           */
/* BEGINNING OF SUB-PROCEDURE QC_HARDCOPY. VERSION 0.                        */
/*                                                                           */
gchart_name='#'.chart_name;
/* Construct the grouphome data chart name. Here a subdirectory is used.    */

type "";
type "Working..";
type "";

if call_flag <> 'S' then
     {
     qc_graph1 = gchart_name.'_qc_chart';
     qc_graph2 = gchart_name.'_qc_chart_stdev';
     if lastcol(gchart_name)>11 then qc_graph1 = qc_graph1.'_means';
/* Here I create two variables that have the core data table name as their   */
/* first letters, and QC_CHART as the next letters. QC_GRAPH1 will be used   */
/* for the name of the INDIVIDUALS QC chart. QC_GRAPH2 will be used for the  */
/* name of the STDEV QC chart (for 'Multiples' data).                        */
/* Adding '_MEANS' to QCGRAPH1 for M type tables is required.                */
/* This construct is duplicated in BQC, so the charts are named the same.   */
/* This section of code is for calls from MAKE... and ADD... procedures.     */
     }
   else
     {
     qc_graph1 = 'SUMMARY_'.chart_name.'_QC_CHART';
     qc_graph2 = 'SUMMARY_'.chart_name.'_QC_CHART_STDEV';
     if lastcol(gchart_name) > 11 then qc_graph1 = qc_graph1.'_MEANS';
     sumry_name= 'SUMMARY_'.chart_name.'_DATA';
/* Construct names for summary chart and graph(s). Used when called from  */
/* SUMMARY...                                                             */
     }

printer='ln03l';
/* Set the PRINTER system variable to the appropriate printer type.         */
/* In this case a DEC LN03 printer in landscape mode.                       */

printout graph(qc_graph1) at (.05,.05) width .9 height .9 on file 'print.tmp';
/* Create a VMS file (print.tmp) in the GROUPHOME (not absolutely necessary) */
/* containing the information RSPRINT needs to create a printable file.     */
/* Note that this file (print.tmp) is not yet printable.                    */
/* The display modifiers slightly shrink the graph and position it with     */
/* margins.                                                                 */

if obj$exists( qc_graph2 ) then
     begin;
     printout graph(qc_graph2) at (.05,.05) width .9 height .9
```

```
            on file 'print.doc';
      chain "append print.doc print.tmp";
 /* Create a second VMS file with the second graph and attach (append) it to */
 /* the first. Same display modifiers.                                       */
      end;

 if yesanswer('Do you need a printout of the data table? ') then
       begin;
       if call_flag <> 'S' then
         printout table(gchart_name) HEIGHT .8 WIDTH .7 AT (.15,.15)
            on file 'print.doc2';
        else
         printout table(sumry_name) on file 'print.doc2'
           HEIGHT .8 WIDTH .7 AT (.15,.15);
 /* The display modifiers here were needed to prevent a wrap-around problem  */
 /* experienced with some LN03's. In those cases the last column of data     */
 /* printed would be wrapped to the first column, and would overwrite the    */
 /* next line of data.                                                       */
       chain 'append print.doc2 print.tmp';
       end;

    chain 'rsprint -o print.six print.tmp';
 /* RSPRINT translates the RS/1 output into something the printer can        */
 /* understand. The -o flag indicates that the next name on the command line */
 /* is the name of the VMS file that will hold the printable data. The last  */
 /* name on the line is the VMS file that has the information RSPRINT will   */
 /* be translating.                                                          */

 get_printer:
 /* The lable GET_PRINTER marks the start of the section that asks the user  */
 /* where printout is to be directed and then prints there.                  */

 type "";
 type "Which printer? Please enter 3 for Sample Receiving.";
 a= getnumber('Number? ');
 /* Asks the user where he wants to print and gets user's reponse.           */

 if a = 1 then
 /* The following code allows me to print in my offic by limiting printout */
 /* to people working under my username (hopefully just me.                */
       begin;
        if (strip($getjobinfo("username")) = 'T8385' ) then
             chain 'copy print.six slkls::txa6:';
         else
             begin;
             type "";
             type "Sorry, that's not installed yet. Please use #2";
             type "";
             goto get_printer;
             end;
      end;

 if a = 2 then chain 'print/nofeed/queue=LN03_704_1T print.six';
 /* If option "2" is selected, printout is directed to C101 */

 if a = 3 then chain 'print/nofeed/queue=LN03_772_T print.six';
```

```
/* If option "3" is selected, printout is directed to 772_t */

chain 'delete print.*;*';
/* Clean up all the files (possible prior versions as well).          */

end;
```

PROCEDURE 8. #SUMMARY_QC_CHART

```
/* SUMMARY_QC_CHART - This procedure creates a either a QA report chart,    */
/* kept permanently in QA records, or a QC chart, for trouble-shooting      */
/* purposes, by modifying display parameters of pre-existing QC charts      */
/* (created via MAKE_QC_CHART or modified by ADD_POINTS_TO_QC_CHART).       */
/*                                                                          */
/* Version 0  Originator: Kirk L. Shanahan  Date: Oct. 6, 1989              */
/*                                                                          */
/* This procedure asks the user to select a QC chart, either from a list    */
/* of 'standard' tests or by entering the test name. It then gives the      */
/* user the option of producing a QA report chart or a QC summary chart.    */
/* Further, it gives the user the option of producing a monthly summary or  */
/* a general summary (which could include the whole chart). Hardcopy of the */
/* chart is automatic. Hardcopy of the data is optional.                    */
/*                                                                          */
/* BEGINNING OF PROCEDURE SUMMARY_QC_CHART. VERSION 0.                      */
/*                                                                          */
erase;
/* Clears the screen.                                                       */
type "";
/* This construct types a blank line to the screen (used for spacing only). */
type "";
type "";
type "Welcome to the QC world. This is procedure SUMMARY_QC_CHART.";
type "It will ask you for the table, month, and year to use, and ";
type "will then make a summary QC chart (or charts).";
type "";
type "";
type "  Version 0.0";
type "";
/* Banner - Lets user know which procedure he/she is in.                    */
crlf=num_to_ascii(13).num_to_ascii(10);
/* CRLF is a text variable containing the non-printable characters <CR><LF>. */
/* It is used below to space out the chart key and notes.                   */
getname:
/* This is a label used as a jump-in point from a later GOTO stmt.          */
/*  It is the starting point of the procedure.                              */
sum_chart_type=gettext(
'Do you want a QA (A) report chart or a QC (C) summary chart?
Please enter an A or a C: ');
/* This is where the user designates which type of summary chart he wants.  */
sum_chart_type= CAPS(sum_chart_type);
/* Capitalize the chart type.                                               */
if not (sum_chart_type = 'A' OR sum_chart_type = 'C' ) then goto getname;
/* Making sure a correct answer was entered.                                */
type "";
IF YESANSWER('Are you summarizing a standard test?') THEN
/* If the user answers "Yes" here, the procedure goes to the CONTROL_LIMITS */
/* table and displays a list of standard tests for the user to choose from. */
/* A "No" sends the procedure to the section that gets a name directly from */
/* the user.                                                                */
 {
type"";
type "A list of methods will now be displayed.";
```

```
type "You should locate the test you want and remember its row number.";
type "You will be asked to enter the row number after the display is finished.";
type "If the test you want doesn't appear, it is not a 'standard' test";
type "In that case, enter a 1 (chooses NON_STANDARD) to go back to the start.";
type "";
checker1: if not yesanswer("Are you ready?") then go to checker1;
/* The TYPE-CHECKER1 block above is meant to give the user a chance to get   */
/* for the upcoming display.                                                 */
   get_test_name:
   DIS COL 1 OF #tmx_CONTROL_LIMITS;
/* The table in this statement is the CONTROL_LIMITS table. Here it is stored */
/* in a grouphome subdirectory and 'personalized' via the 'tmx' designation  */
/* for the Analytical Development Section QC effort.                         */
   control_row= GETNUMBER('Please choose which test by entering the number:');
/* The user enters the row number of the test for which a chart is to be     */
/* constructed.                                                              */
      if control_row=1 then goto getname;
/* The CONTROL_LIMIT table should always have row 1 col 1 be "NON_STANDARD". */
/* This allows the user the chance to go back to the beginning when needed   */
/* via the GOTO above.                                                       */
   dis row control_row of col 1 of #tmx_control_limits;
   if not yesanswer('Is this the test you wanted?') then goto get_test_name;
/* Display the user's choice and confirm it.                                 */
   chart_name = row control_row col 1 of #tmx_control_limits;
   table_type = caps(row control_row col 4 of #tmx_control_limits);
/* Get the stored info CHART_NAME and TABLE_TYPE on the test from the CONTROL */
/* _LIMITS table.                                                            */
   goto make_names;
/* Jump past the next block because it is for a 'non-standard' test.         */
 }
control_row=0;
/* Sets the variable CONTROL_ROW to 0 to avoid problems with an EMPTY value  */
/* later on, set to 0 since to reach here, a 'non-standard' test is implied. */
chart_name =GETTEXT("What is the name of the QC Chart you wish to summarize? ");
/* Chart_name is the variable that holds the core of the table and QC chart */
/* names. GETTEXT is an RPL funtion for terminal I/O that gets a string.     */
/* Here the input string is assigned to 'chart_name'.                        */
make_names:
gchart_name = '#'.chart_name;
/* Create the name of the grouphome table by attaching the #.   */
if not tableexists(gchart_name) then
      begin;
      type "That QC table isn't in the grouphome.";
      type "";
      type "Please check the name and try again.";
      type "";
      goto exiter;
      end;
/* The IF tests to see if the core table is present. If it is, the procedure */
/* goes on. If not, it prints an error message, and goes to the label EXITER */
/* (found at the end of the procedure) which stops the procedure.            */
qc_graph1 = 'SUMMARY_'.chart_name.'_QC_CHART';
qc_graph2 = 'SUMMARY_'.chart_name.'_QC_CHART_STDEV';
if lastcol(gchart_name)>11 then qc_graph1=qc_graph1.'_MEANS';
sumry_name= 'SUMMARY_'.chart_name.'_DATA';
qc_graph3 = gchart_name.'_QC_CHART';
```

```
qc_graph4 = gchart_name.'_QC_CHART_STDEV';
if lastcol(gchart_name)>11 then qc_graph3=qc_graph3.'_MEANS';
msgs_name1= qc_graph1.'@msgs';
msgs_name2= qc_graph2.'@msgs';
/* Construct names for summary chart and graph(s) and the grouphome charts   */
/* graphs.                                                                   */
if obj$exists(qc_graph1) then delete graph(qc_graph1);
if obj$exists(qc_graph2) then delete graph(qc_graph2);
if obj$exists(sumry_name) then delete table(sumry_name);
/* If the objects already exist, then delete them.                           */
if yesanswer('Do you want a simple monthly summary?') then goto month_sum;
/* If the user answers "Yes", the procedure selects all unique month-year    */
/* combinations (NOTE: This assumes all data is in chronological order.),    */
/* and displays them. The user chooses one and the procedure makes the       */
/* chart just for that month's data.                                         */
/* A "No" implies the user wishes to select a contiguous block of data, not  */
/* bounded by just dates, to display.                                        */
c = (lastrow(gchart_name)-60);
/* The variable C is used in the following to limit the display of data      */
/* points to 60 max. It does not limit the selection of summarized data to   */
/* the last 60 points.                                                       */
if c<1 then c=1;
/* Probably unnecessary, unless data table is messed up.                     */
get_rows:
dis rows c to (lastrow(gchart_name)) of col 7 of table(gchart_name);
/* Displays the last 60 points by date only.                                 */
type "";
a= getnumber('Which row should start the summary chart? ');
b= getnumber('Which row should stop the summary chart? ');
/* Asks the user to define the block of data to be summarized by row numbers */
/* in the data table.                                                        */
if a >= b then
      {
      type "";
      type "Start point equal to or greater than stop point.";
      type "Please try again.";
      type "";
      goto get_rows;
      }
/* This IF makes sure that the start point is before the stop point.         */
goto copy_graphs;
/* Skips the following section of code, which does the monthly summary stuff. */

month_sum:
/* The next section of code searches the DATE entries of the data table and  */
/* selects unique month/year combinations. It places the month and year in   */
/* TEMP_DATA and then displays the resultant table. The user then selects    */
/* which row is to be summarized.                                            */
j=1;
/* J is used to indicate the current row of TEMP_DATA in the subsequent      */
/* section.                                                                  */
testmonth = monthpart( row j col 7 of table(gchart_name) );
testyear  = yearpart ( row j col 7 of table(gchart_name) );
/* The functions MONTHPART and YEARPART extract the month and year from a    */
/* DATE. The values are numeric (i.e. "JUN" is not returned, but "6" is).    */
/* These values are stored in TESTMONTH and TESTYEAR.                        */
```

```
row j col 1 of temp_data = testmonth;
row j col 2 of temp_data = testyear;
row j col 3 of temp_data = 1;
j = j + 1.;
/* The first data row is always unique, so the month and year are written   */
/* into TEMP_DATA. The next write will be to the next row, therefore J must */
/* be incremented up by one.                                                */
/* Column 3 of TEMP_DATA stores the row at which that month's data starts.  */
/* Column 4 will store the stop point. It is filled in later.               */
do i =  2 to lastrow( gchart_name);
/* This DO loop selectes each row of the data table and tests the DATE for  */
/* uniqueness. I is the current row in the data table.                      */
 IF ( monthpart( row i col 7 of table(gchart_name) )<> testmonth
    OR yearpart( row i col 7 of table(gchart_name) )<> testyear ) then
/* The condition of this IF is a logical OR. If either the month or year of */
/* the current row of the data table is unique, a new row must be written to */
/* TEMP_DATA.                                                               */
/* Please note that this logic assumes the data table is sequentially time- */
/* ordered.                                                                 */
       begin;
/* The BEGIN marks the start of the section that writes the new unique month */
/* and year to TEMP_DATA, and resets the test variables to the new values.  */
        testmonth = monthpart( row i col 7 of table(gchart_name) );
        testyear  = yearpart ( row i col 7 of table(gchart_name) );
/* Resetting the test variables to the new unique values.                   */
        row j col 1 of temp_data = testmonth;
        row j col 2 of temp_data = testyear;
        row j col 3 of temp_data = i;
        row (j-1) col 4 of temp_data = i-1;
        j = j + 1 ;
/* Writing the new unique values to row J of TEMP_DATA, and then increasing */
/* J by 1. Also writes the stop point in the previous row at column 4.      */
       end;
/* End of the IF BEGIN--END block.                                          */
end;
/* End of the data check DO loop.                                           */
row lastrow('temp_data') col 4 of temp_data = lastrow(gchart_name);
/* Fills in the final stop point for the table.                            */
row 0 col 1 of temp_data='MONTH';
row 0 col 2 of temp_data='YEAR';
/* Labeling the columns of interest in TEMP_DATA.                           */
get_the_row:
/* GET_THE_ROW is the label that marks the start of the user interface for  */
/* obtaining which row represents the data to be summarized.                */
display cols 1,2 of temp_data;
/* Display all the unique month/year combinations.                          */
type "";
type "Please enter the row number of the table above that has";
type "the month and year you wish to summarize.";
month_row= getnumber('Which row number? ');
/* Here the user is requested to enter the row number of TEMP_DATA that     */
/* represents the month the user wishes to summarize.                       */
display row month_row of cols 1,2 of temp_data;
/* Display the selected row for confirmation of choice.                     */
type "";
if not yesanswer('Is this the row you wanted? ') then goto get_the_row;
```

```
/* The procedure asks the user if the choice he made is correct. If it is,  */
/* the procedure goes on, otherwise it goes back to GET_THE_ROW.            */
a = row month_row col 3 of temp_data;
b = row month_row col 4 of temp_data;
/* Save the selected start and stop points in the grouphome data table.    */
type "";
type "OK, I'm working on it now.";
/* A success message.                                                       */
copy_graphs:
type "Working...";
copy graph(qc_graph3) to graph(qc_graph1);
/* Copies the grouphome QC chart to the user's USERHOME, so that the       */
/* grouphome QC chart is not altered.                                       */
set low of x axis of graph(qc_graph1) to a-.25;
set high of x axis of graph(qc_graph1) to b+.75;
/* Set the low and high of the X axis according to the start and stop points */
/* in the underlying data table. Offsets are to allow display of "STD"      */
/* messages.                                                                */
call #qc_y_axis_setup( qc_graph1, a, b);
/* Reset the Y axis low and high for the current display window.            */

if tableexists('temp_data') then delete temp_data;
/* TEMP_DATA is used below, so first it must be deleted.                    */
if sum_chart_type='C' then copy table(msgs_name1) to table('temp_data');
/* Copy the MSGS table of the graph to TEMP_DATA for alteration if the      */
/* requested summary type is for a 'QC' chart.                              */
delete table(msgs_name1);
/* Deletes the MSGS table of the graph. For a QA chart, no further action   */
/* is taken. For a QC chart, the MSGS table is altered to account for the   */
/* display windowing as TEMP_DATA and then renamed.                         */
if sum_chart_type='A' then goto next_step;
/* For a QA chart, the alteration of TEMP_DATA is skipped.                  */

do i= 4 to lastrow('temp_data');
      if (row i col 2 of temp_data >= (a + .5) ) then
        {
        d= i - 1;
        row d col 2 of temp_data = a + .25;
        doexit;
        }
       else d=4;
end;
/* This DO loop locates the message that holds the STD info for the starting */
/* region of the summary chart.                                             */
if d > 4 then  delete rows 4 to (d-1) of temp_data;
/* Deletes "STD" messages that are outside of the display window.           */
rows 1 to 3 of col 2 of temp_data = a+2;
/* Repositions the initial "STD" message to the start of the summary chart. */
rename table temp_data to msgs_name1;
/* Renames TEMP_DATA to the MSGS table of the graph, so that the messages can */
/* be seen.                                                                 */
message=graphnotes of graph(qc_graph1);
/* For a QC chart, the graph notes need to be fixed up to delete any        */
/* reference to previous display windows and to add current display window */
/* information. The variable MESSAGE is used to hold the text during        */
/* alteration.                                                              */
```

```
if message<> empty then
/* The message should never be EMPTY, but just in case...                  */
     {
     if caps(ext(1,1,message))='C' then
/* If the QC chart has been 'windowed' by ADD_POINTS_..., a message will   */
/* have been added starting with the word 'Current'. This line needs to be */
/* deleted if present. The EXT function gets the first letter of the graph */
/* note, then it is capitalized. The IF checks to see if it is a 'C'. If yes */
/* the next section of code deletes the first line of the message.         */
       {
       i=2;
       txt=num_to_ascii(13);
/* TXT will be the new graph note. It is initialized to <CR>.              */
        do while $ext_line(message,i) <> EMPTY;
/* The $EXT_LINE function extracts a line from a text. The line extracted is */
/* given by I. The text is in the variable MESSAGE. 'I' was initialized to  */
/* 2, thereby skipping the line we wished to delete, and is incremented     */
/* below, as long as the extracted line is not EMPTY.                       */
        txt = txt. $ext_line(message,i).crlf;
/* Appends the non-empty line to the variable TXT.                          */
        i = i + 1;
/* Increments I.                                                            */
        end;
       graphnotes of graph(qc_graph1)= txt;
/* Set the graph notes of the QC chart to the new value.                    */
       }
     }
     graphnotes of graph(qc_graph1) =
       'Displaying subgroups '.a.' to '.b.', inclusive.'.crlf.
             graphnotes of graph(qc_graph1);
/* Adds a first line to the graph notes indicating the window subgroup range.*/
next_step:
QCDATA='@'.QC_GRAPH1.'@XTICKS';
/* Construct the name of the XTICK table.                                   */
     if sum_chart_type = 'A' then
       {
       delete table( qcdata ) where col 1 = 'NEW STD';
       call obj$setvalue(qc_graph1,'DISMODS','NOHEADER NONOTES RIGHTKEY BOX');
/* If the summary chart is a 'QA' type, the XTICKS table is deleted and     */
/* the chart's dispay modifiers are set with the OBJ$SETVALUE function.     */
       }
c=ext ( 1, 5, caps ( global('TERMINAL') ) );
/* The GLOBAL fucntion gets an external global variable, in this case the   */
/* system variable TERMINAL. The CAPS capitalizes it. The EXT gets the first */
/* 5 letters, which are stored in variable 'C'.                             */
/* This is done to check if the terminal type implies the use of the RSTERM */
/* terminal emulator. RSTERM does not do date labels on the X axis well, and */
/* the XTICKS display must be suppressed in order to see the graph.         */
/* This suppression and reactivation is done in the subprocedure IBMPC_FIXUP.*/
if c = 'IBMPC' then call #ibmpc_fixup(qcdata, qc_graph1);
 else dis graph(qc_graph1);
/* The call to IBMPC_FIXUP if the terminal-type implies the use of RSTERM.  */
/* The call passes the name of the XTICKS table and the graph itself.       */
/* If RSTERM is not used a simple display is done. IBMPC_FIXUP does the     */
/* display otherwise.                                                       */
if not yesanswer('Is this O.K.? ') then
```

```
/* Confirm that what the user now sees is what he wants. If 'yes' the      */
/* procedure goes on. If 'no' it deletes the summary graph and goes back to */
/* the beginning.                                                          */
   {
   delete graph(qc_graph1);
   goto doanother;
   }
nextgraph:
if obj$exists(qc_graph4) then
/* If a 'STDEV' chart exists it must be modified the same way the 'MEANS'  */
/* chart was modified. This is done in this IF statement.                  */
/* The code contained within {...} is the same as above.                   */
      {
      copy graph(qc_graph4) to graph(qc_graph2);
      set low of x axis of graph(qc_graph2) to a-.25;
      set high of x axis of graph(qc_graph2) to b+.25;
      if sum_chart_type='C' then copy table(msgs_name2) to temp_data;
      delete table(msgs_name2);
      if sum_chart_type='A' then goto next_step2;
      row d col 2 of temp_data = a+.25;
      if d>4 then delete rows 4 to (d-1) of temp_data;
      rows 1 to 3 of col 2 of temp_data = a+2;
      rename table temp_data to msgs_name2;
      next_step2:
      if sum_chart_type = 'A' then
        {
        qcdata = '@'.qc_graph2.'@xticks';
        delete table( qcdata )where col 1 = 'NEW STD';
        call obj$setvalue(qc_graph2, 'DISMODS','NOHEADER NONOTES RIGHTKEY BOX');
        }
      message=graphnotes of graph(qc_graph2);
      if message<> empty then
        {
        if caps(ext(1,1,message))='C' then
                {
                i=2;
                txt=num_to_ascii(13);
                do while $ext_line(message,i) <> EMPTY;
                        txt = txt. $ext_line(message,i).crlf;
                        i = i + 1;
                end;
                graphnotes of graph(qc_graph2)= txt;
                }
        }
      graphnotes of graph(qc_graph2) =
        'Displaying subgroups '.a.' to '.b.', inclusive.'.crlf.
                graphnotes of graph(qc_graph2);
      }
/* End of the code to make a summary 'STDEV' chart.                        */
make table table(sumry_name) from rows a to b of table(gchart_name);
/* Make the summary data table for printout   */
row 0 col 0 of table(sumry_name)='TimeStamp';
row 1 col 0 of table(sumry_name)=date();
row 2 col 0 of table(sumry_name)=time();
/* Place a timestamp in col 0 of the printout table.                       */
call #qc_hardcopy(chart_name, 'S');
```

```
/* Call the hardcopy subprocedure. Here the call flag is set to 'S'. The   */
/* subprocedure uses this in the name-building section.                     */
delete table(sumry_name);
delete graph(qc_graph1);
if obj$exists (qc_graph2) then delete graph(qc_graph2);
/* Deletes the printout tables and graphs from the USERHOME area when hard */
/* copies have been made.                                                   */
/* The OBJ$EXISTS funtion returns a TRUE value if the named object (first in */
/* the parameter list, CHRT1 above) exists. If it does exist, the above     */
/* statement deletes it.  */
/* The DELETE is done only if the graph exists.                             */
doanother:if yesanswer('Do you want to do another? ') then goto getname;
/* Gives the user of restarting the procedure to do another summary.        */
exiter: type "All finished. See you next time.";
/* Ends the procedure with a message.                                       */
```

PROCEDURE 9. #MULTIPLE_COMPONENT_QC_CHART

```
/* This is procedure MULTIPLE_COMPONENT_QC_CHART. It is used to enter data  */
/* from multiple component standards as a block in control charts.          */
/* Originator: Kirk L. Shanahan  2/21/90   Version 0.                       */
erase;
/*  Clear the screen  */
type "";
/* This construct types a blank line. It is used for spacing on the screen.  */
type "";
type "";
type "";
type "";
type "";
type "Welcome to the QC world.";
type "This is procedure MULTIPLT_COMPONENT_QC_CHART (MCC).";
type "MCC will prompt you for data and create or modify a data table.";
type "MCC will then create or modify the appropriate QC chart, and";
type "then display a table showing if all your components are in control.";
type "";
type "  Version 0.0";
type "";
type "";
/* Banner - contains procedure name so user can verify he is in the right  */
/*          one.                                                            */
     a=gettext('Type <Return> to continue.',,true);
/* This is a dummy statement that allows the user to read the banner.       */
if tableexists('temp_data2') then delete temp_data2;
/*  temp_data2 is the temporary data table that this procedure uses for     */
/*  data entry. It eventually is copied to a permanent table in the         */
/*  GROUPHOME.                                                              */
if tableexists('report_table') then delete report_table;
/* REPORT_TABLE is the table that will hold indicators of out-of-control pts. */

  rt_row=0;
/* RT_ROW (abbrev. of report_table_row) is the current row of REPORT_TABLE.  */
  row 0 col 1 of report_table = 'COMPONENT';
  row 0 col 2 of report_table = 'DATE';
  row 0 col 3 of report_table = 'DATA POINT';
  row 0 col 4 of report_table = 'RULE VIOLATED';
  row 0 col 5 of report_table = 'STD. VALUE';
  title of report_table = 'Report Table for Multiple Component QC Charting';
  notes of report_table ='If this table is empty, the test is in control.';
/* Set up the column headers, title, and notes of REPORT_TABLE.             */
get_mcl_col:
dis row 0 of #multiple_component_list;
/* This table holds the component abbreviation list for the multiple        */
/* component standard used. Row 0 holds the accepted overall name for the   */
/* standard. */
mcl_col=getnumber('For which method set will you enter data (Please enter column
 number)? ');
/* Ask the user to choose the standard/method name and enter it.            */
if ( mcl_col <= 0 OR mcl_col > lastcol('#multiple_component_list') ) then
     {
     type "";
```

```
      type "Sorry, you must enter a column number from the displayed list.";
      type "Please try again.";
      a=gettext('Type <Return> to continue.',,true);
      goto get_mcl_col;
      }
/* Check to make sure the entered number is acceptable. If not, try again. */
points_per_line = row 1 col mcl_col of #multiple_component_list;
lc= points_per_line + 10;
/* Get the number of points per line from the master table. Calculate the  */
/* number of columns in TEMP_DATA by adding 10 to it. Store in LC.         */
row 1 col 0 of temp_data2= 'DATE';
do i = 2 to lastrow('#multiple_component_list');
if row i col mcl_col of table('#multiple_component_list')=EMPTY then doexit;
row i col 0 of temp_data2= row i col mcl_col of
                                  table('#multiple_component_list');
end;
/* Once an acceptable standard/method block is chosen, copy the abbreviations */
/* to the temporary table that will be used for data entry. Also add the DATE */
/* row.                                                                       */

row 0 col 1 of temp_data2='Value';
do i= 2 to points_per_line;
row 0 col i of temp_data2='Value'.i;
end;
/* This section of code writes column labels into the temporary data table */
/* TEMP_DATA2. The following WATCH command will only work for pre-exising  */
/* columns.                                                                */
watch col 1 to points_per_line of temp_data2;
/* Set up to watch the data as it is entered.                              */

type "";
row 1 col 1 of temp_data2 = getdate('What is the date for this set? ',,date());
type "";
/* Start by asking for the date of the data set. Assume default is TODAY.  */

     do i = 2 to lastrow('temp_data2');
/* Under a DO LOOP, enter the data for each abbreviation. No EMPTY's allowed. */
     type "";
     c = 'What is the value for '.row i col 0 of temp_data2.'? ';
/* Here I construct the phrase that will appear at the user's terminal from  */
/* a standardized part and a variable part, which is taken from the          */
/* temporary data table.                                                     */
     row i col 1 of temp_data2 = getnumber(c);
/* Ask the question requesting data, and put it in col 1.                    */
      do j =  2 to points_per_line;
/* For multiples-type data, redo the above code with a slightly different   */
/* message, reflecting the fact that we are entering more data.             */
       type "";
       c = 'What is the next value for '.row i col 0 of temp_data2.'? ';
       row i col j of temp_data2 = getnumber(c);
      end;
     end;
data_check:
/* Label indicating start of the data correction section.                    */
if points_per_line>1 then
/* If multiples-type data, I will calculate a CheckSum value, the sum of the */
```

```
/* data in that row, to simplify data entry error detection. If the method   */
/* capable, it will be set up to calculate a CheckSum for comparison to the  */
/* one calculated here. (Not all methods will have a CheckSum available.)    */
     {
     do ii= 1 to lastrow('temp_data2');
/* DO for all the rows in the data table.                                     */
     row ii col (points_per_line+1) of temp_data2= sum of cols 1 to
       points_per_line of row ii of temp_data2;
/* Enter in a new last column the checksum value, calculated via the RS/1    */
/* SUM function.                                                             */
     end;
     row 0 col (points_per_line+1) of temp_data2='CHECKSUM';
/* Label the new column.                                                     */
     }
DIS TEMP_DATA2;
/* Display the data table so that the user can check it over.                */
if yesanswer('Is this all OK? ') then goto data_ok;
/* If the data was entered correctly, skip over the next lines of code.      */
     get_row: a= getnumber('Which row has the bad data? ');
/* Ask the user which row has the bad entry.                                 */
     if a < 1 OR a>lastrow('temp_data2') then
/* Make sure the row number given is a real number and not a mistake.        */
       {
       type"";
       type "That row number doesn't exist. Please try again.";
       type"";
       goto get_row;
/* If the entered number is wrong, tell the user and give him another chance. */
       }
     get_col: b= getnumber('Which column has the bad data?');
/* Ask the user which column has the bad entry.                              */
     if b < 1 OR b > lastcol('temp_data2') then
/* Make sure the column number given is a real number and not a mistake.     */
       {
       type"";
       type "That column number doesn't exist. Please try again.";
       type"";
       goto get_col;
/* If the entered number is wrong, tell the user and give him another chance. */
       }
     if (a=1) then c=getdate('What is the correct date? ');
      else
       c= getnumber('What should I put there instead? ');
/* Get the 'correct' value.                                                  */
     row a col b of temp_data2 = c;
/* Go and correct the value.                                                 */

if points_per_line>1 then delete column 'CHECKSUM' of temp_data2;
/* If multiples-type data, the CheckSum data column must be deleted or it    */
/* will be propagated multiple times at the end of TEMP_DATA2.               */
     goto data_check;
/* Go back to the beginning, redisplay the table, etc., etc.                 */

data_ok:
if tableexists('temp_data') then delete temp_data;
allocate table temp_data 1 row by 11 cols;
```

```
/* The block of data will now be transformed into individual entries to      */
/* separate GROUPHOME QC charts and data tables. The table TEMP_DATA is used */
/* by the other QC procedures for holding data before it is put into the     */
/* GROUPHOME. Therefore, here I create that table, and fill it up below under */
/* a DO loop for each component of the multi-component standard.             */

do j= 2 to lastrow('temp_data2');
/* MASTER CHART BUILDING LOOP                                                */
/* Starts at 2 because row 1 holds the date. DO for each row.                */
     do k= 1 to lastrow('#tmx_control_limits');
     if row j col 0 of temp_data2 =
       row k col 0 of #tmx_control_limits then doexit;
     end;
/* This short DO loop searches the CONTROL_LIMITS table for the row that    */
/* corresponds to the element shown in the data table. The logic here assumes */
/* a match will be made.                                                     */

     ROW 1 COL 4 OF TEMP_DATA = ROW K COL 7 OF
              TABLE('#tmx_control_limits');
/* Set col 4 of TEMP_DATA equal to the standard's value.                     */
     ROW 1 COL 5 OF TEMP_DATA = ROW K COL 2 OF
              TABLE('#tmx_control_limits');
/* Set col 5 of TEMP_DATA equal to the standard's control limit value.       */
     ROW 1 COL 6 OF TEMP_DATA = ROW K COL 8 OF
              TABLE('#tmx_control_limits');
/* Set col 6 of TEMP_DATA equal to the standard reference's value.           */
     ROW 1 COL 7 OF TEMP_DATA = ROW 1 COL 1 OF TEMP_DATA2;
/* Set col 7 of TEMP_DATA equal to the DATE from the data table.             */
     ROW 1 COL 8 OF TEMP_DATA = ROW j COL 1 OF TEMP_DATA2;
/* Set col 4 of TEMP_DATA equal to the datum from the data table             */
     if LC>11 then
/* LC>11 when multiples-type data. Implies we need to move more data over. */
       {
       do i = 2 to points_per_line;
       ROW 1 COL (i+7) OF TEMP_DATA = ROW j COL i OF TEMP_DATA2;
       end;
/* Move the additional data from the data table to TEMP_DATA.                */
       ROW 1 COL 1 OF TEMP_DATA = mean of cols 8 to (7+points_per_line)
                                  of row 1 of temp_data;
       ROW 1 COL 2 OF TEMP_DATA = stdev of cols 8 to (7+points_per_line)
                                  of row 1 of temp_data;
       ROW 1 COL 3 OF TEMP_DATA = points_per_line;
/* Fill in colums 1 to 3 of TEMP_DATA with the mean, st. dev., and count    */
/* info that the procedure needs.                                            */
       }
     ROW 1 COL (lc-2) OF TEMP_DATA = DATE();
     ROW 1 COL (lc-1) OF TEMP_DATA = TIME();
     ROW 1 COL (lc  ) OF TEMP_DATA = STRIP($GETJOBINFO("USERNAME"));
/* Fill in the time and ID stamp information.                                */
     qctable = '#'.row K col 1 of
                       TABLE('#tmx_control_limits');
/* Construct the QC table name from the method name.                         */
/* Note that QCTABLE is different from QC_TABLE (used below in the CALL to */
/* to BUILD_QC_CHARTS) because of the '#' prefix. To construct the table */
/* now I need the prefix, while BUILD_QC_CHARTS adds it itself.              */
     table_type = row K col 4 of TABLE('#tmx_control_limits');
```

```
/* Get the TABLE_TYPE variable that BUILD_QC_CHARTS needs.                */
     stdvalue = row 1 col 4 of temp_data;
/* Get the STDVALUE variable that BUILD_QC_CHARTS needs.                  */
     control_limit = row 1 col 5 of temp_data;
/* Get the CONTROL_LIMIT variable that BUILD_QC_CHARTS needs.             */
     call_flag='A';
/* Set the CALL_FLAG variable to 'A' (to indicate adding points).         */
   MOVE_TO_GRPHOME:
/* Label indicating start of code that copies the data to the grouphome and */
/* calls the chart building procedures.                                   */
     IF OBJSEXISTS(QCTABLE) THEN
/* Check to make sure the data table exists in the grouphome.             */
       {
       ADD ROW TO TABLE(QCTABLE) FROM ROW 1 OF TEMP_DATA;
/* If the data table is in the grouphome, add the new data to it.         */
       }
     ELSE
       {
       call_flag='M';
       row 0 col 1 of table(qctable) = 'MEAN';
       row 0 col 2 of table(qctable) = 'ST. DEV.';
       row 0 col 3 of table(qctable) = 'COUNT';
       row 0 col 4 of table(qctable) = 'STD. VALUE';
       row 0 col 5 of table(qctable) = 'CONTROL LIMIT';
       row 0 col 6 of table(qctable) = 'STD. REF.';
       row 0 col 7 of table(qctable) = 'DATE';
       row 0 col 8 of table(qctable) = 'VALUE';
       if lc>11 then
              {
              do jj= 2 to points_per_line;
              row 0 col (7+jj) of table(qctable) = 'VALUE'.jj;
              end;
              }
       row 0 col (lc-2) of table(qctable) = 'ENTDATE';
       row 0 col (lc-1) of table(qctable) = 'ENTTIME';
       row 0 col lc of table(qctable) = 'OPERATOR';
/* If the table doesn't exist in the grouphome, neither should the chart.  */
/* Therefore the CALL_FLAG variable is reset to 'M' (indicating making a new */
/* chart). Then the grouphome data table is created by filling in all the  */
/* column labels.                                                         */
       GOTO MOVE_TO_GRPHOME;
/* Once the table has been created the procedure goes back to the code above */
/* and adds a row to it, just as if it had been there before. Note however, */
/* that the CALL_FLAG variable's value is now different.                  */
       }
     qc_table = row K col 1 of TABLE('#tmx_control_limits');
/* Get the un-prefixed gouphome data table name for the CALL statement next. */
     CALL #BUILD_QC_CHARTS(QC_TABLE, table_type, call_flag, true,k, -1);
/* The CALL to BUILD_QC_CHARTS. The -1 parameter at the end of the list tells */
/* BQC to suppress the display.                                           */

rt_row= rt_row+1;
/* RT_ROW started at 0 above, now we are ready to check for out-of-control */
/* flags, so we increment RT_ROW by 1. Any out-of-control flags will result */
/* in an entry being written to this row of REPORT_TABLE. No flags means a */
/* blank row in REPORT_TABLE.                                             */
```

```
if table_type = 'M' then
      {
      chart_name1 = '#'.qc_table.'_qc_chart_means@data';
      chart_name2 = '#'.qc_table.'_qc_chart_stdev@data';
      }
 else
      chart_name1 = '#'.qc_table.'_qc_chart@data';
/* Create the appropriate QC chart names for the current data row.        */
ii = lastrow(chart_name1);
/* Get the new last row of the data table underlying the QC chart.        */
      if row ii col 9 of table(chart_name1) <> empty then
/* If column 9 is not empty, an out-of-control flag is present.           */
        {
        row rt_row col 1 of report_table = row j col 0 of temp_data2;
        row rt_row col 2 of report_table = row ii col 0 of table(chart_name1);
        row rt_row col 3 of report_table = row ii col 2 of table(chart_name1);
        row rt_row col 4 of report_table =
                       row ii col "RULE #" of table(chart_name1);
        row rt_row col 5 of report_table = stdvalue.' +/- '.control_limit.'%';
/* When an out-of-control flag is present, I write the component's        */
/* abbreviation into column 1 of REPORT_TABLE, the data point's DATE into */
/* column 2, the data point's VALUE into col 3, the SPC rule violated into */
/* col 4, and a message indicating what the standard value and control limit */
/* were for the data point into col 5.                                    */
        }
if table_type = 'M' then
/* If the chart is for multiples-type data, the standard deviation control */
/* chart must also be checked.                                            */
      {
      rt_row = rt_row +1;
/* Increment the row in REPORT_TABLE for the new chart.                   */
      if row ii col 9 of table(chart_name2) <> empty then
        {
        row rt_row col 1 of report_table = row j col 0 of temp_data2;
        row rt_row col 2 of report_table = row ii col 0 of table(chart_name2);
        row rt_row col 3 of report_table = row ii col 2 of table(chart_name2);
        row rt_row col 4 of report_table =
               row ii col "RULE #" of table(chart_name2);
        row rt_row col 5 of report_table = control_limit.'%';
/* Same logic as above except it is now done on the STDEV control chart. The */
/* message in column 5 is a little different to reflect that.             */
        }
      }
END;
del temp_data;
del temp_data2;
/* Clean up by deleting the temporary data tables. */
dis report_table;
checker2: if not yesanswer('Finished?') then goto checker2;
/* Display the report table for the user and wait till he is finished     */
/* (indicated by answering 'YES' to the question) before proceding.       */
exiter: type "All finished. See you next time.";
/* a final message to the user indicating the procedure is done.          */
```

PROCEDURE 10. #QC_Y_AXIS_SETUP

```
procedure( qcgraph, xlow, xhigh);
/* This is subprocedure QC_Y_AXIS_SETUP. It's purpose is to adjust the      */
/* QC graph Y axis low and high to just include all necessary points, lines, */
/* and other information. It works from a 'window' specification given by   */
/* XLOW and XHIGH in the arguments. QCGRAPH is the passed name of the graph. */
/*                                                                          */
/* Originator: Kirk L. Shanahan  6/21/90                                    */
/* Version 0.0                                                              */
/*                                                                          */
data_table=qcgraph.'@data';
/* Construct the underlying data table's name.                              */
/*                                                                          */
/* The next section of code tries to determine what the Y axis high should be */
/* based on the data and control limits.                                     */
/*                                                                          */
/* RS/1 does not always fill in the control limit value columns of the data */
/* table if the data has been entered in blocks. This will cause a problem  */
/* if the ends of the display window happen to fall where no data is. The   */
/* subsequent logic will fail due to EMPTY values. Therefore I check for a  */
/* value in the lowest display window row in the control limit columns, and */
/* if they are not found, I go back till I find one and fill it in at the   */
/* low end of the display window.                                           */
    if row xlow col 'UCL' of table(data_table)=empty then
/* The IF statement that checks for an empty control limit value at the     */
/* lowest point of the display window. (The highest will always have an     */
/* entry.)                                                                  */
      {
      do loop = xlow to 1 by -1;
/* This do loop counts down from the point just before the display window   */
/* start till it finds a value (or it reaches the first row, which should   */
/* always have a value in the control limit columns).                       */
        if row loop col 'UCL' of table(data_table) <> empty then
/* Check this row for a non-empty value. If not found, do the next row back. */
          {
          row xlow col 'UCL' of table(data_table) =
            row loop col 'UCL' of table(data_table);
          row xlow col 'CL' of table(data_table) =
            row loop col 'CL' of table(data_table);
          row xlow col 'LCL' of table(data_table) =
            row loop col 'LCL' of table(data_table);
          doexit;
/* When a non-empty value is found, set the display window lows equal to the */
/* contents of this row (row LOOP), and exit the DO loop.                   */
/* The 'CL' column value is set because the NEXT time this procedure is     */
/* the DELTA calculation far above will crash unless a value is in col 4.   */
          }
        end;
/* End of the DO loop.                                    */
    }
/* End of the IF statement used to fill in the control limit columns.       */
/* Now we are ready to find max and min of Y values.                        */
    a= maximum of rows xlow to xhigh
            of col 2 of table(data_table);
```

```
/* Start with the maximum data point inside the display window.          */
   b = maximum of rows xlow to xhigh of col 'UCL' of
            table(data_table);
/* Get the highest control limit within display window.                  */
   if a>b then b=a;
/* Use B to store the maximum. Here, replace it by A if A is larger.    */
      c=.1;
      if b<1 then c=.01*b;
/* Now, define C to be the offset added to B (the Y high) for clarity of  */
/* display. If b>1, an offset of .1 units is used, but this is too large for */
/* b<1. It squeezes the plot down. Therefore, check if b<1 and reset C to  */
/* 1% of B when b is small.                                              */
   HIGH OF Y AXIS OF GRAPH(qcgraph)= b + c;
/* Set the high to the value determined above, with a little extra so that  */
/* the line isn't drawn on top of the BOX of the graph.                  */
   yhigh = b + c;
/* Save the new Y axis high for later use.                               */
/* The next section of code attempts to set the QC chart's Y axis low value. */
/* This requires looking at the data, the lower control limit line, and the  */
/* 'STD=' messages.                                                      */
      ylow = .97* minimum of rows xlow to xhigh
               of col 2 of table(data_table);
/* Start with the minimum data point inside the display window. (The .97 is */
/* for display offset.)                                                  */
   ylow2 = minimum of rows xlow to xhigh of col 'LCL' of
            table(data_table);
/* Get the lowest control limit inside the display window.               */
      if  ylow2 <ylow then ylow= ylow2;
/* select between current y low or lowest data point (inc. new data) */
      if ( (chart_type='SA') and (ylow < 0) ) then ylow=0;
/* check ylow for the "SA" case and adjust if needed  */
      yheight= 1.07*(yhigh - ylow)/8.25;
/* 8.25 is #" height of graph on paper, 1.07 = 1/2 of " of 'STD' msg.   */
      if ( (ylow <> ylow2) AND (yhigh-ylow>1) ) then
                        ylow = ylow - 2.5 * yheight ;
/* if ylow equals old ylow then don't need to adjust ylow for msg spacing     */
/* if not equal, need to lower ylow to allow for message space. Also, if the  */
/* total Y axis height is < 1, the messages space adjustment done in the IF   */
/* will mess up the display, thus it is done only for >1 height.              */

      LOW OF Y AXIS OF GRAPH(qcgraph)= ylow;
/* Set the Y axis low.                                                   */
end;
```

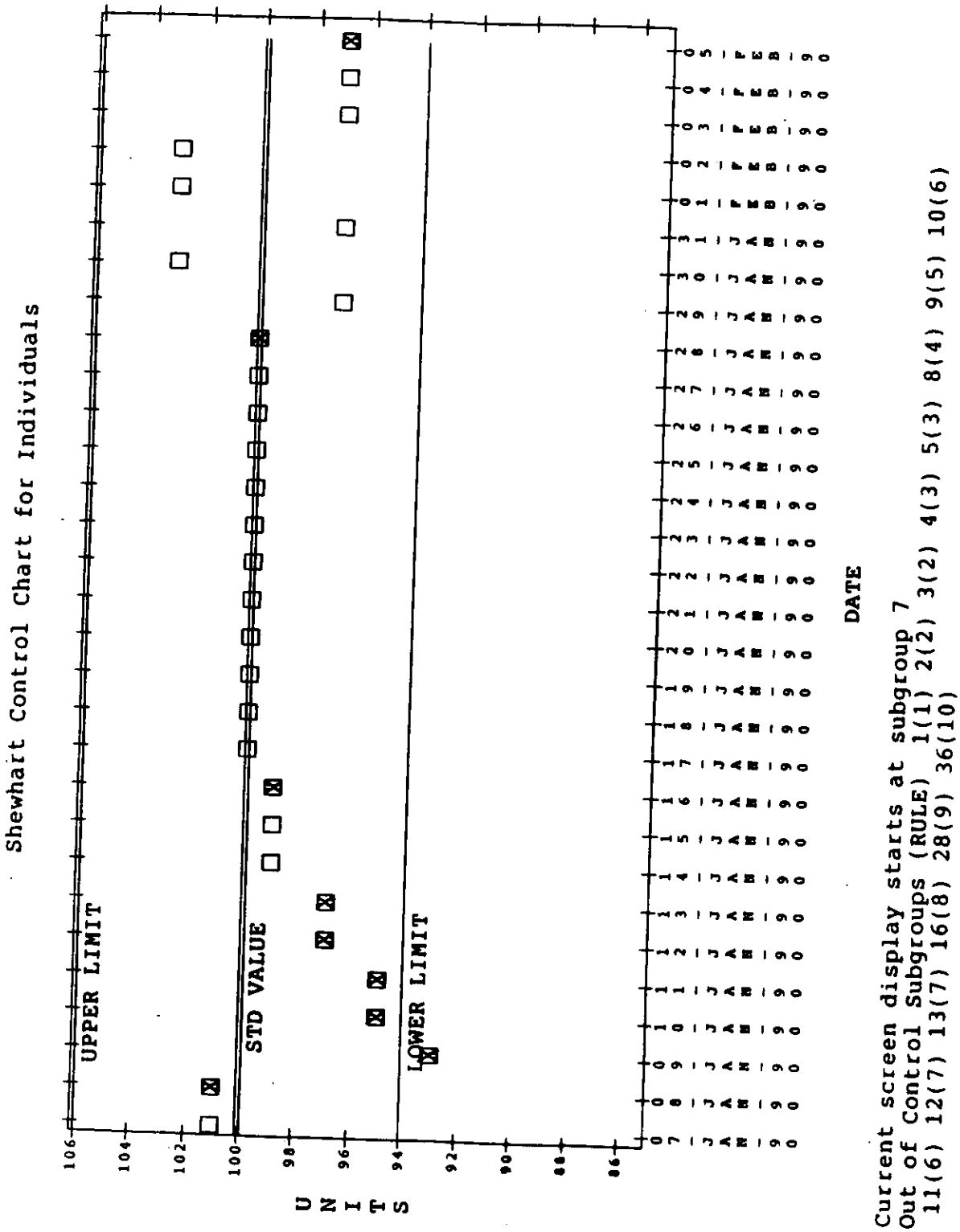

Figure AI-1 Screen Display of a Control Chart Illustrating Windowing of Display to 30 Points (QC Format)

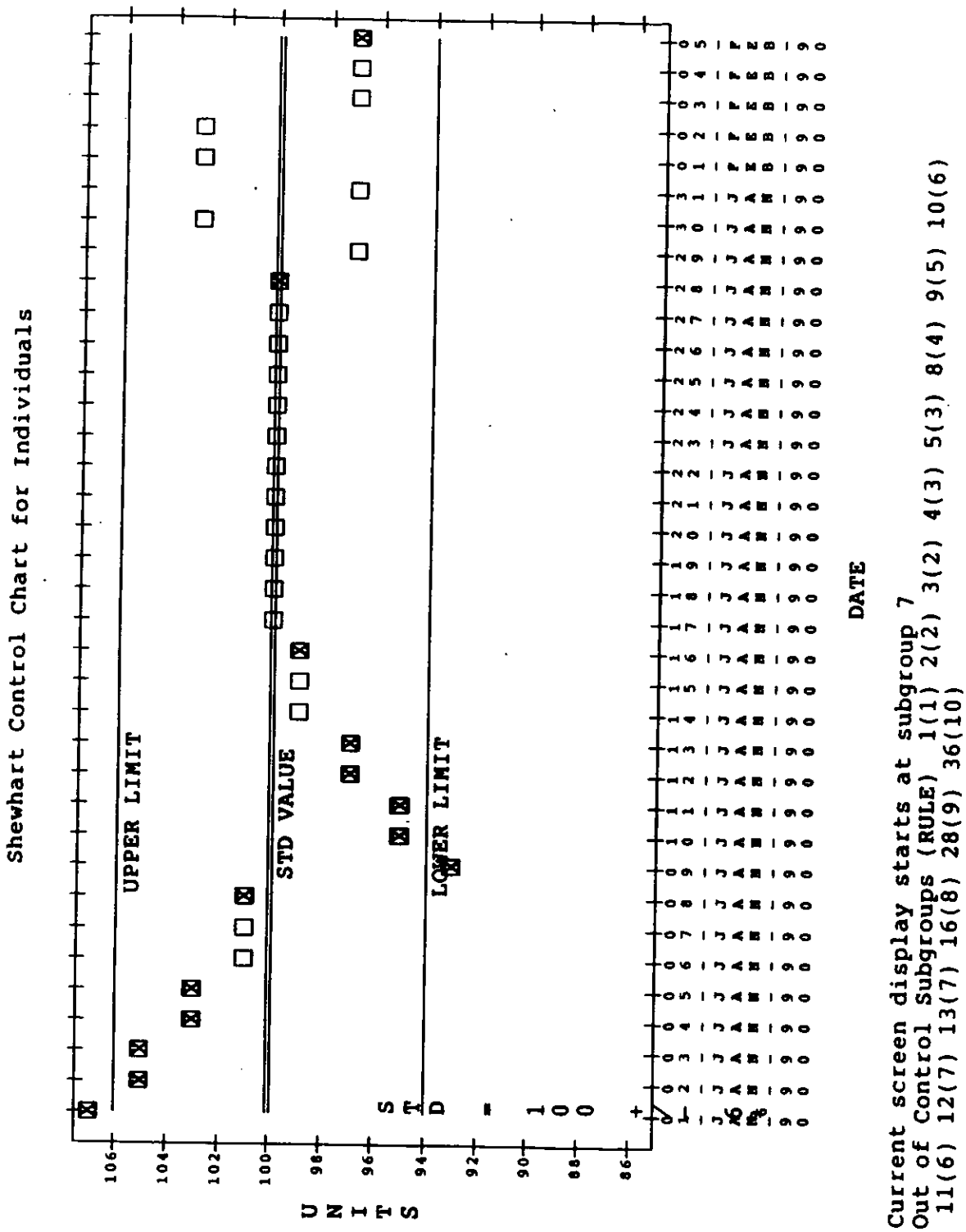

Figure AI-2 Full Individuals Test Data Control Chart (QC Format)

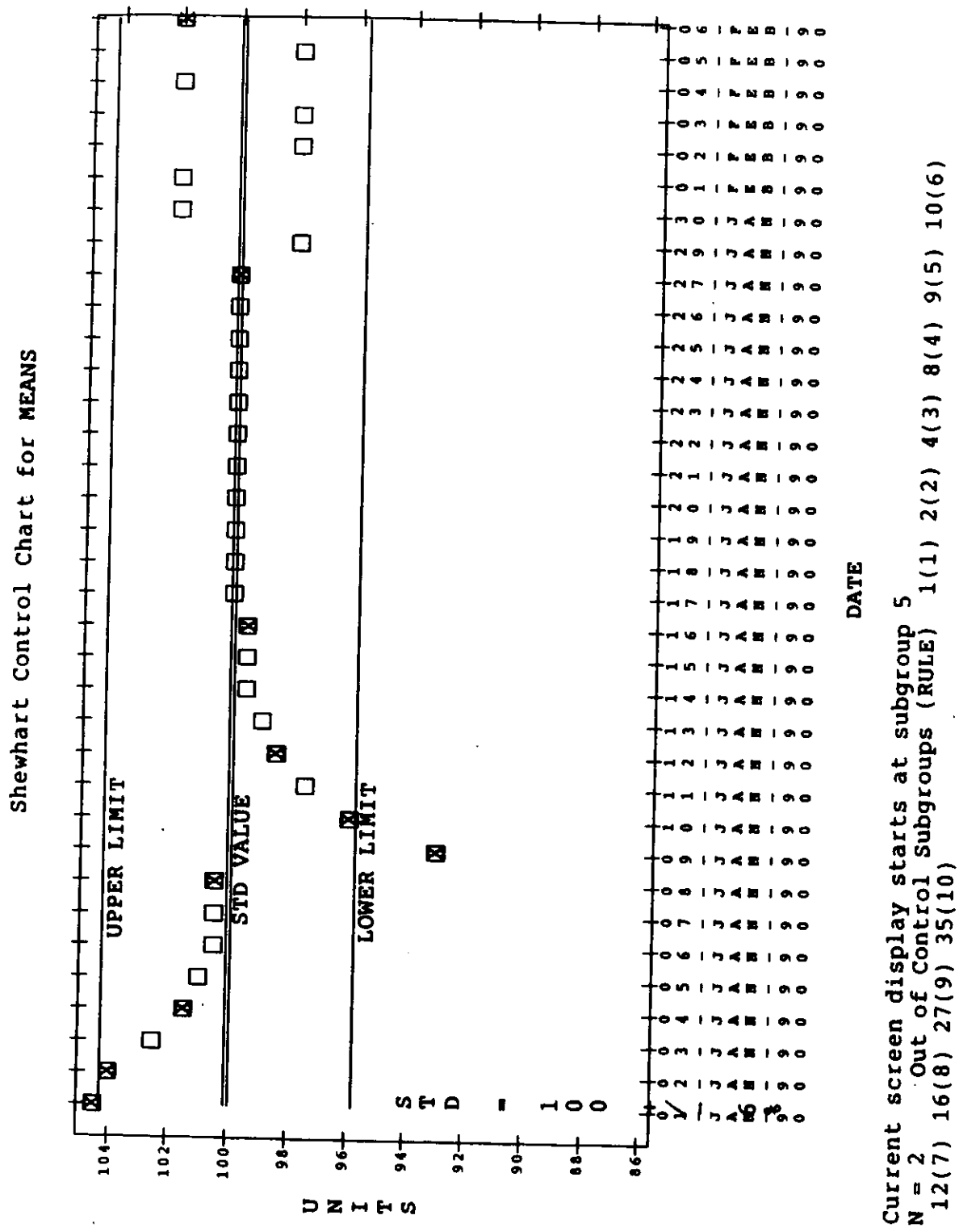

Figure AI-3 Full Multiples Test Data Means Control Chart (QC Format)

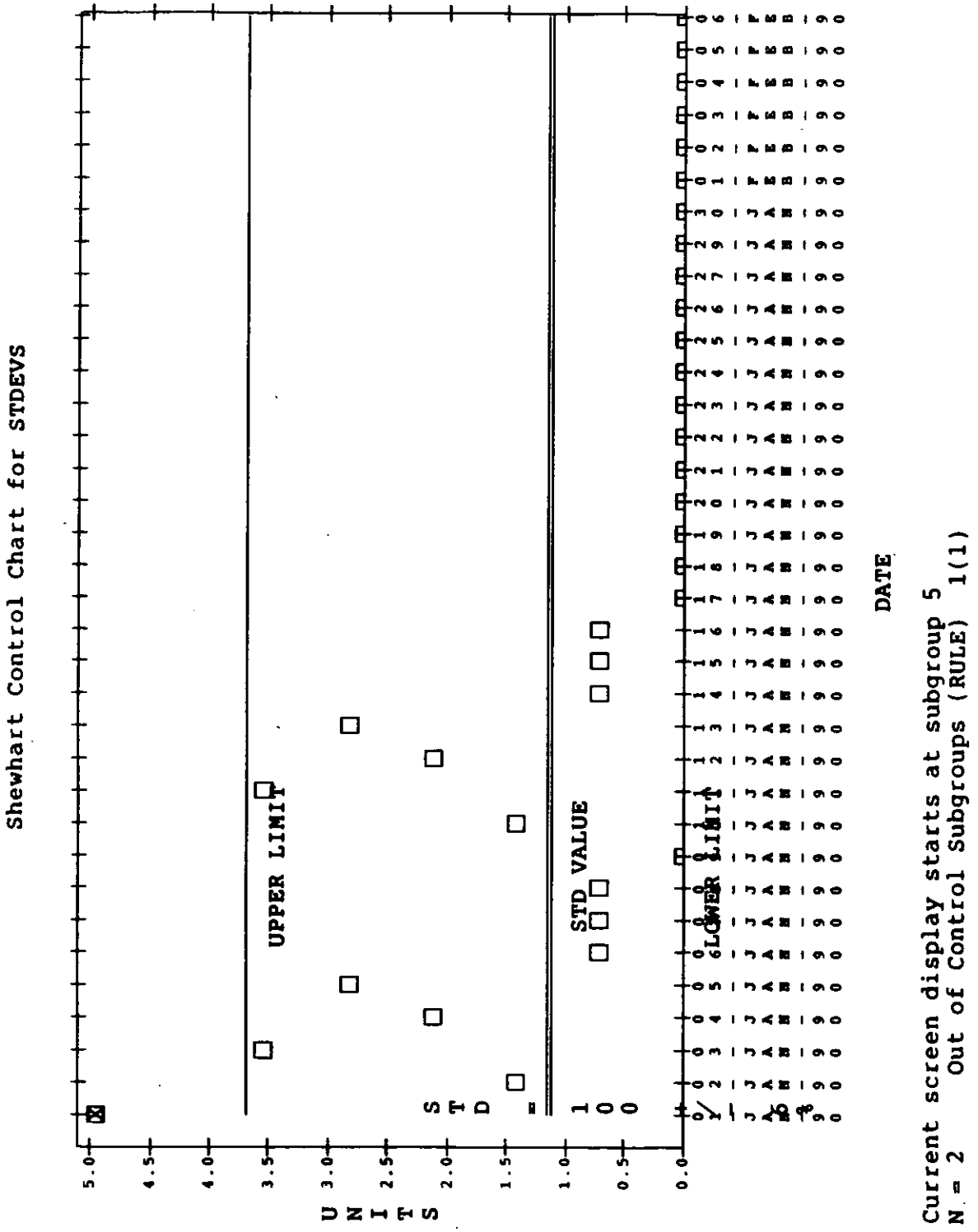

Figure AI-4 Full Multiples Test Data Standard Deviations Control Chart (QC Format)

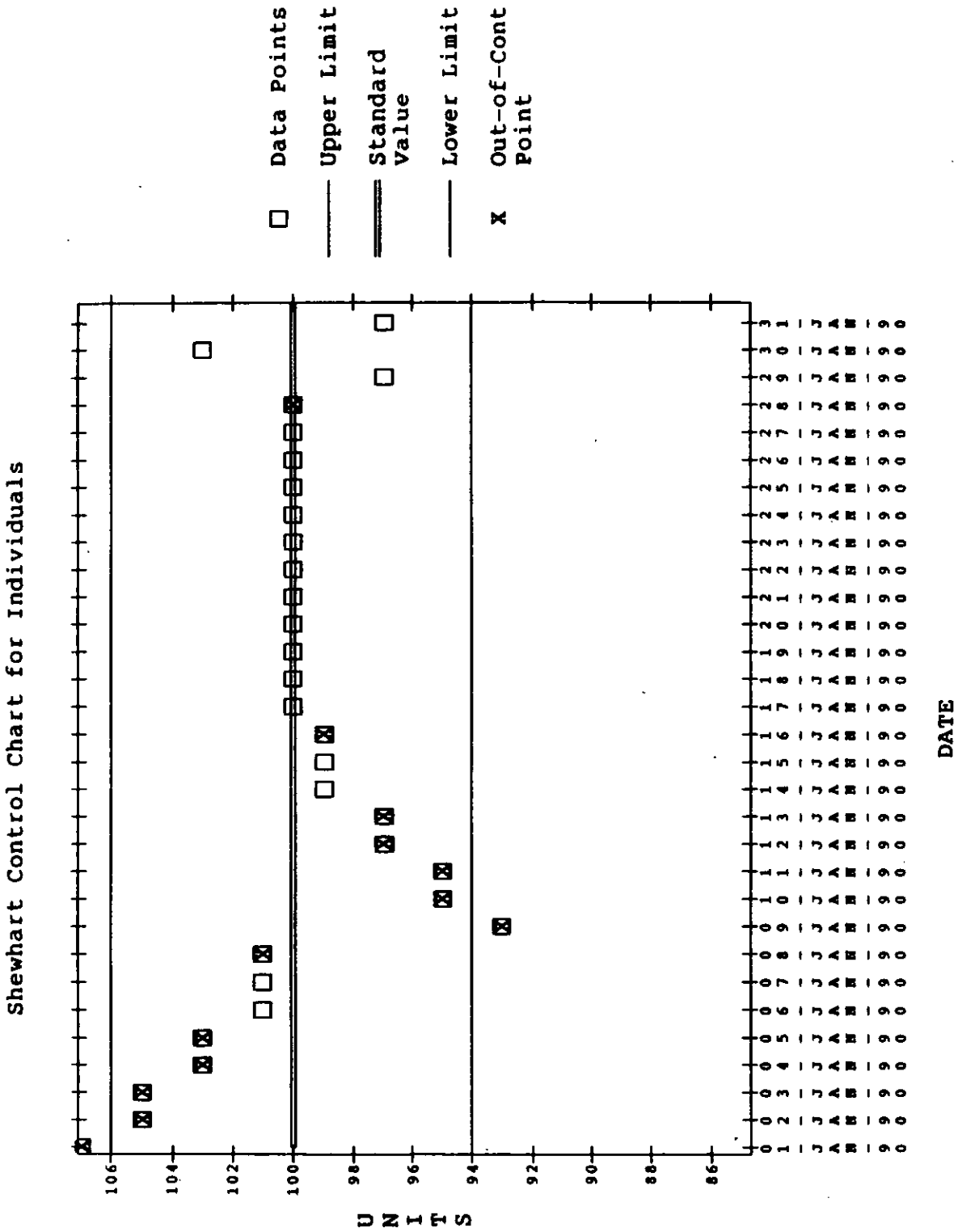

Figure AI-5 Monthly Summary Printout of Individuals Test Data Control Chart QA Format)

TABLE AI-1.
#QCMENU
Table used by the QC menu procedure as the top-level entry point.

0 CHOICE	1 TASK
1 MAKE a New Chart	call #make_qc_chart
2 ADD POINTS to an Old Chart	call #add_points_to_qc_chart
3 GENERATE a Summary Chart	call #summary_qc_chart
4 DO MULTIPLE COMPONENT QC Charts	call #multiple_component_qc_chart
5 LOG OUT of RS/1	LOGOUT
6 EXIT to the # sign	EXIT

The first column appears on the screen when the user enters 'call #menu' at the # prompt. The second column contains the command to be executed when the appropriate choice is made.

TABLE AI-2.
TEMP_DATA
The temporary table used in data entry.

0 KLS_TEST_I	1 MEAN	2 STDEV	3 COUNT	4 STDVALUE	5 RANGE %OFSTD	6 STD REF
1				100	6	REF #1
2				95	5	REF #2

0	7 DATE	8 VALUE	9 ENTDATE	10 ENTTIME	11 OPERATOR
1	06-JUL-90	100	07-JUL-90	15:55:43	SHANAHAN
2	07-JUL-90	94	07-JUL-90	15:55:43	SHANAHAN

The user sees columns 4 to 8 only. All columns are copied to the grouphome table before TEMP_DATA is deleted. This example has two data points, each with it's own standard.

TABLE AI-3.
TEMP_ROWS
The table used by BQC to define regions in the new data.

0	1	2	3	4	5	6
1	100	6	1	1	STD=100 +/- 6%	73.2
2	95	5	2	2	STD=95 +/- 5%	73.2

Column 1 is the standard's value, Column 2 is the standard's control limit, column 3 is the start point of the region in TEMP_DATA (the row number), Column 4 is the stop point of the region, Column 5 holds the message to be added to the control chart when a new standard is entered, and Column 6 holds the Y position of the message. A separate line will exist for each unique region of the newly entered data.

TABLE AI-4.
TEMP_DATA2
The temporary table used for data entry in
MULTIPLE_COMPONENT_QC_CHART

0	1 Value1	2 Value2	3 CHECKSUM
DATE	07-JUL-90		07-JUL-90
B	100	101	201
Ca	50	48	98
Fe	23	22	45

This example shows the table produced on the screen for a 3 component standard run in duplicate, where B, Ca, and Fe designate the components. Each row is used along with the date to generate a 1-row TEMP_DATA table for each component. The CHECKSUM column is provided for those users that have a row sum available as an aid in detection of data entry errors.

TABLE AI-5.
REPORT_TABLE
Screen display from MULTIPLE_COMPONENT_QC_CHART that indicates any out-of-control indicators.

Report Table for Multiple Component QC Charting

0	1 COMPONENT	2 DATE	3 DATA POINT	4 RULE VIOLATED	5 STD. VALUE
1	M	01-AUG-90	85.000000	5	100 +/- 5%
2	M	01-AUG-90	7.071068	1	5%
3	M2	01-AUG-90	62.500000	1	55 +/- 2%
4	M2	01-AUG-90	17.677670	1	2%

If this table is empty, the test is in control.

This example indicates four out-of-control indicators, two based on the data mean, and two based on the data standard deviation, for two different standards.

Appendix II. Software User Manual

Included in this Appendix is a sample user's manual distributed to DWPF laboratory personnel during initial training in the use of these procedures. As such, it reflects the installation at that time. Each installation will probably be different in details, especially those concerned with hard copy printing.

Appendix III. Setup Instruction for 'Standard' Methods

Setting up a 'standard' method requires a basic understanding of RS/1 table editing. The relevant table is hard-coded into the source code, but typically resides in the laboratory's grouphome and its name includes the words 'control limits'. For example in the case of the TNX DWPT laboratory, the table is #TNX_CONTROL_LIMITS. This table is shown in Table AIII-1.

TABLE AIII-1.

#TNX_CONTROL_LIMITS 13R x 8C 01-AUG-90 14:33 Page 1

Control Limits and other information on TNX Standard Tests

0	1 METHOD	2 CONTROL LIMITS %	3 UNITS	4 TABLE TYPE
1	NON_STANDARD			
2	Kirk_test_I	2	GARBONZOES	i
3 M	Kirk_test_M	5	BOINGOS	M
4 M2	Kirk_TEST_M2	2	bongos	m
5 CHO2	CHO2	5	PPM	I
6 NH4	NH4	5	PPM	I
7 CL	CL	5	PPM	I
8 FL	FL	5	PPM	I
9 C2O4	C2O4	5	PPM	I
10 NO2	NO2	5	PPM	I
11 NO3	NO3	5	PPM	I
12 PO4	PO4	5	PPM	I
13 SO4	SO4	5	PPM	I

0	5 CONTROL CHART TITLE	6 POINTS/LINE	7 Std Value	8 Std Ref
1				
2	This is a Shewhart Contr			
3	This ia another junk cha	2	100	3
4	whoops!	2	55	aaa
5	Shewhart Control Chart f		10	
6	Shewhart Control Chart f		20	
7	Shewhart Control Chart f		2	
8	Shewhart Control Chart f		1	
9	Shewhart Control Chart f		10	
10	Shewhart Control Chart f		10	
11	Shewhart Control Chart f		15	
12	Shewhart Control Chart f		10	
13	Shewhart Control Chart f		5	

This table has eight columns and thirteen rows. Column 1 contains the accepted method name. Column 2 has the control limit (as a percent of the target value). Column 3 has the units of measurement, which are transposed to the label of the Y axis of the control chart. Column 4 has the table type, a

variable that tells the RS/1 procedure if the method requires multiple data points per determination. 'I' implies 1 point/line and 'M' implies several points/line. Column 5 contains a chart title. This is clipped in Table AIII-1, but can easily be up to 80 characters wide. Column 6 holds the number of points per line if the method is of the type that uses more than 1. Column 7 holds the standard's value. Column 8 holds a reference to where (in a laboratory notebook) the current standard was prepared or analyzed.

To set up a 'standard' method, a row must be added to the control limits table holding all the above information for that method. To use a 'standard' method in a multiple component standard method, Column 0 of the control limits table must hold a mnemonic which will also appear in the multiple component method list table, and which the technician will see when entering data. This list table's name is also hardcoded into the procedure and is typically #MULTIPLE_COMPONENT_LIST. An example from the TNX DWPT grouphome is shown in table AIII-2.

To actually do the row addition, the RS/1 Table Editor can be used. Using the /EXPAND command in the Editor allows the user to expand the table with the cursor motion. Once a row has been added in this manner, the /EXPAND function should be disabled and then the row contents can be entered.

Table AIII-2.

#MULTIPLE_COMPONENT_LIST 8R x 3C 01-AUG-90 14:51 Page 1

MASTER TABLE FOR MULTIPLE COMPONENT STANDARD
QC CHARTING
LIST OF BLOCKS

0	1 IC_CHO2	2 IC_FL	3 TEST_M
1	1	1	2
2	CHO2	FL	M
3	CL	CL	M2
4	NO2	NO2	
5	NO3	NO3	
6	PO4	PO4	
7	SO4	SO4	
8	C2O4	C2O4	

Table AIII-2 shows entries for three different multiple component standard methods. The first two (IC_CHO2 and IC_FL) are real methods for ion chromatographic determination of formate (1) or fluoride (2), chloride, nitrite, nitrate, phosphate, sulfate, and oxalate ion. The third method is an artificial test method.

Both of the IC methods assume 'Individuals' type of charts,

i.e. one determination per point. The test method (3) assumes 'Multiple' type data and 2 points/line. This information is entered on line 1 under the appropriate composite method. At this time, composite methods with mixed data types are not allowed.

On subsequent lines, the mnemonic for the component is entered. The technician will choose the composite method based on the entry in row 0 of this table, and will fill in data in a table next to the mnemonic entered in the body of the list table.

To create a composite method, a MULTIPLE_COMPONENT_LIST table must be constructed (usually in the grouphome) that conforms to the above example. The individual components must be listed as 'standard' methods in the CONTROL_LIMITS table, and a mnemonic must be entered in column 0 of that table that matches one in the list in the MULTIPLE_COMPONENT_LIST table.

Appendix IV. Instructions for Procedure Modification

This Appendix is included to serve as a guide to programmers who wish to modify the RPL code documented in this report. There are some subtleties that need to be explicitly stated.

In the VAX implementations of this code, the procedures, tables, and charts are stored in the grouphome. Changing the procedures requires editing the grouphome procedures, but this is not a straightforward process. RS/1 protects grouphome procedures to a certain extent. In an IBM PC implementation, the use of a grouphome is only indicated if one wishes to use floppy disks to store all the grouphome data. In most instances, the procedures will be 'normal' RPL procedures, meaning that a simple EDIT command will suffice to change the procedure.

For safety's sake, the programmer must always make and save a copy of the old version of the code. The usual RPL editing process does not do this, and thus if a change is detrimental, the old version will have been lost. It will then be impossible to restore the old version without reference to hard copy source code.

To make a copy of a grouphome procedure, an RS/1 procedure must be used. That procedure is $GET_GROUP_PROCS. The programmer types 'call $get_group_procs' at the #, and enters an interactive dialog that prompts him for the name of the grouphome procedure to copy and the name of the userhome procedure that will receive it. This name should be unique, perhaps with a version number included in the name. If the userhome procedure already exists, $GET_GROUP_PROCS asks if the programmer wishes to replace it. The name of the grouphome procedure must be preceded by a # character.

Once a userhome copy of the procedure exists, the programmer must save it in a second procedure in case it is needed later to restore the old version. To do this the programmer must create a new procedure that is exactly the same as the one just created from the grouphome. The programmer could use $GET_GROUP_PROCS a second time to create the historical copy, or he can use a special RS/1 variable-type called procedure definitions. To use the second method, the programmer enters 'DEF OF new_proc_name = DEF OF old_proc_name' at the #. This automatically creates a compiled copy of the 'old' procedure and gives it the specified name (new_proc_name).

Either the 'old' or the 'new' copy can serve as the historical record copy. The programmer should note in a notebook or some other permanent device the copy's name.

Now a simple EDIT command (EDIT proc_name) will place the programmer into the word processing editor with the source code of the procedure. The programmer should now edit the code and document any changes with comments. Properly exiting the word processor will cause RS/1 to save the new version of the

code. RS/1 will then ask if the programmer wishes to compile the code. The code will not execute unless it is compiled.

Testing the code can be accomplished in two fashions. First the new code can be placed in the grouphome and tested by normal use procedures. This method uses a function $MAKE_GROUP_PROCS, described below. It also causes the old grouphome version to be lost. If the programmer is unsure his changes will work, he should utilize the more complicated second method.

The second test method consists of placing all relevant procedures in the programmer's userhome, and editing them to eliminate the use of grouphome procedures. Instead, they should use the userhome versions. This is more complicated, as the programmer usually will have to copy several procedures, and edit each to eliminate calls to grouphome procedures. But it is safer, as the original grouphome procedure will be untouched until a tested new version replaces it.

To replace a grouphome procedure with a new one stored in the userhome, the programmer calls $MAKE_GROUP_PROCS. This procedure also uses an interactive dialog, asking for the userhome procedure name and the grouphome procedure name to copy it to. If the grouphome procedure already exist, $MAKE_GROUP_PROCS asks if the programmer wishes to replace it. With $MAKE_GROUP_PROCS, the grouphome procedure should NOT have a # character at the front.

One potential problem a programmer may encounter when editing grouphome procedures occurs when other users are using any one of the grouphome procedures. In this instance the grouphome table that holds procedures is 'locked', and noone can write to it. Thus a programmer attempting to copy a new version to the grouphome will see a failure message stating that the grouphome procedures table is 'locked due to other users'. The only solution to this is to wait until noone else is using the grouphome procedures table. This is also why the use of the RS/1 procedure $EDIT_GROUP_PROCS is not recommended. $EDIT_GROUP_PROCS locks the procedures table and other users cannot then use any group procedure.

Other RS/1 grouphome management procedures exist and the programmer is referred to the RS/1 manuals for their descriptions. In general, if the instructions given above are used, the other procedures are of minimal use.

To get hardcopy source code, the RPL code must be transferred to a standard ASCII file. The RS/1 procedure $PUTFILE is used for this. The programmer types "$PUTFILE(DEF OF proc_name, 'VMSfilename')" at the #. (Note the quotes around the VMS filename.) The source code is then placed in the named VMS file. The reverse process is accomplished with the $GETFILE procedure. In this case the DEFINITION variable type must be used. The command is: "DEF OF proc_name = $GETFILE('VMSfilename')". This procedure must them be compiled manually

("COMPILE proc_name").

The ASCII file containing the source code can be edited as a normal file. It can be sent via VAXMAIL or incorporated into an All-in-1 mail message using the <GOLD>G function. (Select the VMS menu option. The complete VMS filename should be used. This is: COMPUTER"AcctName Password"::DISK:[DIRECTORY] filename;version#.)