CHAPTER 11

PROCESSING TRANSACTIONS

PROCESSING TRANSACTIONS

INTAC has two facilities for entering, changing, or deleting data records: TR and EDIT. When you have simple data editing requirements (for example, a small number of operations to perform on one INTAC file), the EDIT facility is immediate and easy (see Chapter 7). The TR facility offers much more power and flexibility. Although TR initially requires a three-step process, once you have created a TR program, you can accomplish even complex operations easily and quickly.

The system you set up with TR (for TRansaction) can automate the updating process so that a user needs no knowledge of INTAC or of file structure. TR allows simultaneous operations upon multiple files, use of more than one data item as an external reference, record audit trails, and batch updating. Use TR to customize your interactive data operations by specifying your own prompts. Set up a system that interrogates the data on the file and performs operations depending upon the values in the file. Or let INTAC calculate additional data items when data is entered.

TR can be used to make changes throughout one or more files based upon your criteria; for example, a TR might delete records throughout files ASSET, VENDOR, and ORDER if the purchase order is 10754.

Using TR, you can easily create screen formatting to highlight entries during an interactive session at a video terminal. Screen formatting simplifies data operations and makes the system more attractive to users.

To use TR, you create a program definition file using the Ross Editor or other text editor. This definition file contains your specifications for calculations. The next step is to use the TR facility to translate your definition into a BASIC program. Once you have completed these two steps, anyone can update the file(s) time after time by simply running the program.

TR

Below is an example of the custom prompting possible with the TR facility. Note that the prompting updates two files in the Tri-City asset system (ORDER and VENDOR). (The file definition to produce this prompting is explained later in the chapter.) Since it is not possible to print dynamic video effects, the example shows an update session for a hardcopy terminal. However, this chapter will explain how you can give yourself a video demonstration using a case study file definition.

```
RUN TRQENH

TRANSACTION PROGRAM TRQENH

Enter purchase order number ? 10066
Vendor number? 11
Vendor #11 not on VENDOR.INT
Do you wish to add to file <NO>? YES
Vendor name? THE LIGHTHOUSE
Street address <>? 765 MAPLE
City and state <>? GENEVA, IL
Zip code <>? 60105
```

```
Order and vendor added to respective files
Enter purchase order number ?
Processing complete. Have a nice day
```

ORGANIZATION OF THIS CHAPTER

This chapter is organized into these sections:

The chapter has been written to enable non-programmers to write two kinds of TR definition files. Part 1 explains the overall process and the reason for the two different kinds of files. Part 2 contains information common to both kinds of files: line number rules, documenting the file. Parts 3 and 4 begin with explanations of uses of the type of skeleton, and fully-annotated, simple examples of TR definition files. Also included are examples that perform more complex operations. An orderly procedure for planning the TR is recommended. All file definition sections and commands are explained and illustrated.

Part 5 explains the simple steps to translate the file definition into BASIC program(s) and to run the programs. Use the quick reference pages at the end of the chapter as you plan your own file definitions.

HOW TO USE TR

The TR process consists of these three steps:

1. Create a transaction definition file.

   Using a text editor, create a definition file using .DEF as an extension. This line-numbered file is specially formatted, and contains all the information needed to process/update the data. Follow the instructions in this chapter for the file definition appropriate to your needs. Save the file for use in step 2.

2. Generate a transaction program.

   INTAC validates the statements in the definition file and then merges them with a partial program called a "skeleton" to form one or more complete programs. INTAC allows a choice of two different skeletons according to your needs. (Use

Skeleton B when you need to sort the file records before performing operations on selected records. Use Skeleton Q to create a series of questions to be answered for every record. See the following explanation of the two kinds of processing.)

To initiate this step, enter INTAC, give the TR command, and follow the TR dialogue as shown in this chapter. You only perform steps 1 and 2 one time. If you make changes to the file definitions (see Chapter 5, MO), or to the transaction program definition file, you may need to repeat step 2.

3. Run the generated program.

   Whenever you wish to update the file, you can now simply run the BASIC program(s), using the appropriate system command. (You specify the names of these programs in the file definition.)

TWO KINDS OF PROCESSING

Often you want to set up an interactive updating system that allows interrogation of the data in the file(s) record by record before updating operations are performed. You want answers to a series of questions for each record and then you want to add, delete, or change the record. At other times, you want to go through the file(s) to select certain records, sort them, and then perform operations on only the selected records.

TR has different program skeletons for these two different kinds of processing: Skeleton Q for question and answer prompting and updating; and Skeleton B for sorting and updating records.

The internal operation of INTAC is significantly different for the two different needs. For SKELETON Q, the TR command generates one BASIC program. For SKELETON B, the TR command creates two programs or program segments. The first segment will SELECT and SORT the data from files specified in the SELECT section of the definition file. The second segment will then process the sorted and screened data as specified in the LOGIC section.

The file definitions for these two different skeletons use similar sections and statements, but they differ in many details. In this chapter, the two skeletons are discussed separately.

OVERVIEW OF TR DEFINITION FILES

A TR definition file is divided into sections that must appear in a specific order. Skeleton Q may use these sections: OPTIONS, DEFINE, FILES, LOGIC, and CODE. Skeleton B may use these sections: OPTIONS, DEFINE, FILES, SELECT, LOGIC, and CODE.

For each kind of skeleton, this chapter recommends an orderly procedure for planning your definition file. The last pages of this chapter supply reference pages to guide planning. When you are ready to create the definition file, use a text file editor such as the Ross

Editor. Give it an appropriate filename using .DEF as the extension. When you begin step 2 to generate the BASIC program(s) within the TR facility, you will type in this name in response to a prompt.

FILENAME EXAMPLE: `ASSET.DEF`

LINE NUMBER RULES

The definition file is a line-numbered file. Line numbers are used by the TR facility to keep sections of the file in a necessary order and to control the order of execution of statements within two of these sections. Ross Systems recommends that you begin sections with line numbers as indicated throughout this chapter (and on the reference pages) to make the file easy to read. (Only two sections of a file definition actually require line numbers within a specific range. These are LOGIC 3500-19999 and CODE 20000-23999.) After creating the transaction definition file in the Ross Editor, save it with line numbers.

MULTIPLE STATEMENTS ON ONE LINE

INTAC allows you to place more than one statement on a line in a TR program. A "line" is the code associated with a line number. A "subline" is a physical line separated from the previous physical line by ampersand carriage-return. A "statement" may be an INTAC or BASIC statement.

Each INTAC or BASIC statement should be on its own subline. Between multiple statements on one line, use a backslash (\) at the beginning of the subline.

INTAC does not allow statement modifiers after INTAC statements. That is, you CANNOT have a statement such as `5050 ADD FILE1 IF DEPT.NO >2001`

Any INTAC file statement to be included on a multi-statement line must be the last statement on the line.

EXAMPLE:

```
5040  IF DEPT.NO>4001 THEN &                          carriage-return
      DEPT.TOTAL=DEPT.TOTAL + SALES.TOTAL &          carriage-return
      \SHOW DEPT.TOTAL:COL5 &                        carriage-return
      \SKIP1
```

ERROR HANDLING

Those INTAC LOGIC statements that access INTAC files (for example, ADD a record) make use of an INTAC variable named ERROR% to indicate the results of the operation. Your TR program can check the value of ERROR% and take appropriate action based on the value. INTAC statements that make use of ERROR% are ADD, DELETE, FIND, FINDNEXT, UPDATE, and UPDATI.

Your TR program should reset ERROR% to zero if you want processing to continue for a record for which an error is encountered. Note the use of ERROR% in chapter examples. See Appendix B for more detail about error handling, including the values possible for ERROR%.

VIDEO EFFECTS

INTAC statements allow you to position prompts and text on the video screen, underline or highlight user entries, and create dynamic video effects such as clearing the screen between questions. To use these statements, you must have a VT100 compatible terminal and you must specify TERMINAL VT100 in the file definition OPTIONS section. You include the appropriate video INTAC statements in the LOGIC section.

Occasionally, you will want to run a video program temporarily on a hardcopy terminal. A file definition that uses screen formatting will run on a hardcopy terminal; however, terminal displays will be preceded by apparently meaningless characters as the program attempts to do something with the video statements.

In this chapter, video statements are illustrated and explained most fully in the part on Skeleton Q. These statements are also included in the part on Skeleton B, but without a full-scale example.

DOCUMENTING THE FILE DEFINITION

Place comments directly in your file definition to make it self-explanatory for future users. Ross Systems recommends use of banner headings (such as those shown on the chapter worksheets), as well as comments placed throughout the file.

Precede comments (including any lines of asterisks) with an exclamation point ! Comments can be placed on the same line as a BASIC statement, but not on the same line with an INTAC TR statement. For this reason, comments are best placed on lines by themselves and are easiest to read if highlighted by lines of asterisks.

EXAMPLES:

```
11 !**********************************************************************
12 !TR SKELETON Q FILENAME: TRQSIM.DEF
13 !PURPOSE OF TR: ADD PURCHASE.ORD.NO AND VENDOR.NO TO CASE STUDY FILES
14 !AUTHOR: S. HOPE
15 !DATE: MAY 13, 1982
16 !**********************************************************************

5090                 !***********************
5095                 !*** BEGIN PROCESSING***
5096                 !***********************
```

SKELETON Q: QUESTION FORMAT

Skeleton Q is used for updating one or more files concurrently and selectively. It generates a single program that prompts the user for data items to add, change, or delete specific records.

The system you set up with Skeleton Q can automate the updating process so that a user needs no knowledge of file structure or even of INTAC. You can use this skeleton to customize your interactive data operations by specifying your own prompts. The system can interrogate the data on the file and perform operations based on the values in the file. You can let INTAC calculate additional data items (for example, items based on data in two different files) when data is entered.

This part of the chapter shows how to create special formatting to highlight entries during an interactive session at a video terminal.

SUMMARY OF SKELETON Q SECTIONS

A file definition using SKELETON Q may include sections as shown on the following chart. The section name is on a line by itself and is followed by information appropriate to that section. Line numbers must keep the sections in the order shown on the chart below.

SECTION NAMES	DESCRIPTION OF PURPOSE	PAGE
OPTIONS	Specifies various processing options, such as data validation, terminal type, and skeleton. By default, Skeleton Q will be used; therefore, this section is optional for the Skeleton Q.	11-16
DEFINE	Defines temporary data items used by the transaction program. These are data items not found on any of the INTAC files. (This section is optional.)	11-18
FILES	Specifies the INTAC files to be used in the generated program. If any of these files specify external files, these external files also must be listed in this section.	11--20
LOGIC	Specifies the processing rules for the various questions. This section has a question loop format.	11-21

CODE	Reserved for user BASIC subroutines and functions used for processing the data. (This section is optional.)

11-30

PLANNING THE Q SKELETON TR

As you write a transaction definition file, we recommend that you write sections in an order that reflects your planning process: LOGIC, CODE, FILES, DEFINE, OPTIONS. Assign line numbers to keep the file in the order necessary for the TR facility.

In planning the LOGIC section, first write out the dialogue and decide upon spacing that will make the display easy to read. You can usually estimate the horizontal position to provide spacing; however, you may want to use a formatting ruler. If you are using a VT100 terminal, write down appropriate video effects, such as clearing the line or screen, highlighting, vertical spacing.

Now label the questions (Q10, Q20, etc.). It is convenient to number questions by ten to allow for future addition of questions. Work out any special editing for questions (for example, validation against another file).

Once the LOGIC section is planned, it is easy to fill in the FILES, DEFINE, and OPTIONS sections.

THREE EXAMPLES OF Q SKELETONS

Three versions of essentially the same Q skeleton are included in this chapter. These examples have been designed to illustrate some of the advantages of TR over the EDIT facility and to give you an idea of the creative uses that can be made of TR statements.

First is a definition to add a purchase order number and a vendor number to the Tri-City case study ORDER file. This definition creates a system even easier than the EDIT command for a novice because it controls the process completely.

Following this example is an enhanced version of the file definition that updates the VENDOR file as well as the ORDER file. The third version adds video effects. For each of the first two definitions, the chapter includes the file definition, an explanation, and a print-out of the resulting updating session. For the video definition, the chapter shows the file definition, explains the commands, and gives instructions for running a demonstration at your video terminal.

EXAMPLE 1: Q SKELETON

The Tri-City planner has written a TR that sets up a session to add two data item values (PURCHASE.ORD.NO and VENDOR.NO) to the ORDER file. VENDOR.NO is validated by reference to the external file VENDOR. (These two files are described in Chapter 2.) The

TR file definition consists essentially of some simple custom prompts, defined with the PROMPT statement. Note the use of comments following exclamation marks. Note also the use of the ampersand to continue statements to sublines.

```
11  !***********************************************************************
12  !TR SKELETON Q FILENAME: TRQSIM.DEF
13  !PURPOSE OF TR: ADD PURCHASE.ORD.NO AND VENDOR.NO TO ORDER FILE
14  !AUTHOR: S. HOPE
15  !DATE: MAY 13, 1982
16  !***********************************************************************
17  !
100     OPTIONS
110         SKELETON Q
300     FILES
310         FILE1 ORDER
320         EXTERNAL VENDOR
3500     LOGIC
5000         BEGIN PROGRAM
6000         Q10
6010                  PROMPT PURCHASE.ORD.NO "Enter purchase order number " &
                      EXIT " "
6020                  FIND FILE1 INDEX1 PURCHASE.ORD.NO ERROR=Q20 NOMESSAGE
6030                  SHOW "Sorry, purchase order already in ORDER.INT"
6040                  GOTO Q10
7000         Q20
7010                  ERROR% = 0%     !clear FIND error from Q10 &

7020                  PROMPT F1.VENDOR.NO "Vendor number" &

7030                  ADD FILE1
7040                          !All done; repeat for next purchase order
7050                  SHOW "Order entered"
7060                  GOTO Q10
19000         END PROGRAM
```

EXPLANATION OF FIRST Q FILE DEFINITION

The purpose of this file definition is to add purchase order numbers and vendor numbers to the ORDER file. It will make the process easier than using the EDIT facility chiefly because a session will be initiated and controlled through one simple command. Even temporary employees with no knowledge of INTAC will be able to run the update.

The program prompts for purchase order number. If a purchase order number is a duplicate of one already in the file, the user will be given an error message immediately and prompted again for purchase order number. If the purchase order number is new, then the user is prompted for vendor number.

Q10
The first question sequence begins at line 6000. (The program author uses questions incremented by 10 to make it easy to add questions later if desired. In line 6010, the statement PROMPT defines the prompt for the purchase order number. If the user enters a carriage return, the PROMPT option EXIT transfers control to the end of the program to terminate the session. Note the use of the ampersand to continue line 6010 so that the entire PROMPT statement (with its EXIT option) occupies only one line number.

Lines 6020 and 6030 define error handling for purchase order numbers that are already on the ORDER file. (In this system, a duplicate is an error since purchase order numbers are unique.) The FIND command sends INTAC to the ORDER file (FILE1 in the FILES section at line 310) to retrieve any existing duplicate record for the PURCHASE.ORD.NO data item. If a duplicate record is found, the program continues to line 6030 to print an appropriate message (using the SHOW statement) and then on to 6040 to prompt over again for purchase order number. Note: if the duplicate was not checked here, it would result in an error condition when ADDed at line 7030.

Most of the time in INTAC, a FIND statement is expected to find a record. INTAC considers it an error and prints an error message if no record is found. It also sets an error flag (ERROR%) to a value that identifies the error. However, in this system, the normal mode is to add a new number and to reject any number that already exists. If the purchase order number is not found on the file in line 6020, the user's response is a new number and should be added. Line 6020 uses an ERROR statement to send the program on to the next question to add the record. To avoid confusion, the NOMESSAGE statement suppresses any INTAC error messages that would normally print.

Q20
The second question sequence begins at line 7000. Note that the program moves to this question sequence when no record has been found on the file. INTAC has set an error flag to a code number. Line 7010 is necessary to reset the INTAC error flag to zero. Now the error flag is available in case another error is made. (The INTAC error codes are explained in Appendix B.)

Line 7020 uses the PROMPT statement to cause prompting for vendor number. Note use of the qualified data item name because VENDOR.NO is the item name on both files and the prompt is for the VENDOR.NO to be entered in the ORDER record. Also note that because of the file definition, the VENDOR.NO entered will be checked in VENDOR.INT.

At line 7030, the statement ADD FILE1 causes INTAC to update the ORDER file by adding the record containing two data item values. Line 7050 displays a message through use of the SHOW statement. The program returns to the first question to prompt for another purchase order number.

No provision is made in this TR for error handling if the user makes a typing error for VENDOR.NO or enters a VENDOR.NO not present on the VENDOR file. The usual INTAC error messages for invalid data would be displayed.

Two lines in this TR Skeleton Q are not required, but are recommended as good practice: line 5000 BEGIN PROGRAM and line 19000 END PROGRAM. If you do include these lines, you must include the word PROGRAM.

Note that the case study ORDER file contains only the two data items added by this TR program. A TR file definition must add values for all file data items. If a data item is omitted, random characters (often called "garbage") will be placed in the file when the program is run.

THE RESULTING UPDATING SESSION

The TR program is generated by entering INTAC and using the TR facility. Then the record containing the two data item values can be added to the ORDER file by giving the command RUN TRQSIM at the systems level. Below is a sample update session.

```
RUN TRQSIM

TRANSACTION PROGRAM TRQSIM

Enter purchase order number ? 10053
Vendor number? 10
Order entered
Enter purchase order number ?

END OF TRANSACTION PROGRAM TRQSIM
```

EXAMPLE 2: ENHANCED Q SKELETON TO UPDATE TWO FILES

This second, enhanced example demonstrates additional TR statements. Explanation of these additional statements follows the file definition. Note in this file the use of a backslash for multiple statements on one line number.

```
11 !*****************************************************************
12 !TR SKELETON Q FILENAME: TRQENH.DEF
13 !PURPOSE OF TR: ADD PURCHASE.ORD.NO AND VENDOR INFO TO CASE STUDY FILES
14 !AUTHOR: S HOPE
15 !DATE: MAY 13, 1982
16 !*****************************************************************
17 !
100   OPTIONS
110        SKELETON Q
200   DEFINE
210        YES.OR.NO S 3
300   FILES
310       FILE1 ORDER
320       FILE2 VENDOR
3500   LOGIC
5000       BEGIN PROGRAM
6000         Q10
6010            PROMPT PURCHASE.ORD.NO "Enter purchase order number " &
                EXIT " "
6020            FIND FILE1 INDEX1 PURCHASE.ORD.NO ERROR=Q20 NOMESSAGE
6030            SHOW "Sorry, purchase order already in ORDER.INT"
6040            GOTO Q10
7000         Q20
7010            ERROR% = 0%   !clear FIND error from Q10 &

7020            !input vendor, but do not check vendor file &
                !(note STORE will skip external file check) &

7030            PROMPT F1.VENDOR.NO "Vendor number" STORE &


7040       &
                !look up vendor # on VENDOR.INT
7050            FIND FILE2 INDEX1 F1.VENDOR.NO ERROR=Q30 NOMESSAGE
7060            !vendor found on file, must be OK
7070            ADD FILE1
7080            !all done, repeat for next purchase order
```

TR

```
7100          SHOW "Order entered"
7110          GOTO Q10
8000       Q30
8010          ERROR% = 0%     !clear FIND error from Q20 &

8020          SHOW "Vendor #" F1.VENDOR.NO " not on VENDOR.INT"
8030          PROMPT YES.OR.NO "Do you wish to add to file" DEFAULT "NO" &

8040          IF YES.OR.NO <>"YES" AND &
              YES.OR.NO <>"NO" THEN &
              SHOW "Enter  YES  or NO  please" &
              \GOTO Q30 &

8050          IF YES.OR.NO = "NO" THEN &
              GOTO Q20 &

8060          F2.VENDOR.NO = F1.VENDOR.NO  !assign vendor # &

9000       Q40
9010          PROMPT VENDOR.NAME "Vendor name" &

10000      Q50
10010         PROMPT STREET.ADDRESS "Street address"
11000      Q60
11010         PROMPT CITY.STATE "City and state"
12000      Q70
12010         PROMPT ZIP.CODE "Zip code"
12020                 !add record to order file
12030         ADD FILE1
12040                 !add record to vendor file
12050         ADD FILE2
12060                 !report adds and goto prompt for next order
12070         SHOW "     Order and vendor added to respective files"
12080         GOTO Q10
19000      END PROGRAM
19010         SHOW "     Processing complete. Have a nice day"
```

EXPLANATION OF ENHANCED FILE DEFINITION

The purpose of file definition TRQENH is to add purchase order numbers and vendor numbers to the ORDER file and to add vendor information (including vendor number, vendor name, address) to the VENDOR file whenever a new vendor is used.

Q10 and Q20
The LOGIC section of the file definition is just like the first TR explained earlier up to line 7030, the prompt for VENDOR.NO. In this PROMPT statement, STORE is used to turn off the usual INTAC validation against the edit parameters defined in the file definition. The author handles customized validation and error messages in lines 7050 through 8020. The STORE command stores the user's vendor number response so that the program can use it in following steps.

The FIND statement in line 7050 retrieves from the VENDOR file any vendor number that matches the stored vendor number. When a matching vendor number is found, the program moves on to line 7070 to add the purchase order number and the vendor number to the ORDER file (ADD FILE1), and on to line 7100 to report the ADD. Then the program returns to question 10 to prompt for another purchase order number. (If an error occurs on the ADD, INTAC prints a message and terminates the program.)

If a matching vendor number is not found at line 7050, the ERROR statement sends the program to question 30 without printing the usual INTAC error messages.

Q30
Question 30 (at line 8000) resets the INTAC error flag to zero. Line 8020 uses the SHOW command to print a specific message about the new vendor number. Text messages to be printed are enclosed in quotes. When the program is run, the qualified data item F1.VENDOR.NO will contain the value entered by the user for vendor number.

Also part of the Q30 sequence are statements to let the user choose whether or not to add information to the VENDOR file. Line 8030 prompts for a new data item named YES.OR.NO (note that it is included in the DEFINE section because it is used in a PROMPT statement.) The user response NO is specified as the default in the same line. Line 8040 traps any user response other than YES or NO. Notice that line 8040 consists of four sublines that contain the TR statements SHOW and GOTO. This is an example of a multiple statement line which uses the backslash before the second statement.

If the user enters NO, line 8050 returns to reprompt for vendor number. If the user wants to enter vendor information, a YES response sends the program to line 8060. This line takes the new vendor number previously entered by the user and stored by INTAC (at line 7030) and assigns its value to the data item VENDOR.NO in the VENDOR file (FILE2).

Q40 to Q70
The rest of the questions (40 through 70) prompt for values to be added to the VENDOR file. Note that all data items in the file receive values. Line 12050 performs the add of values to the five items in the VENDOR file. Note: For simplicity, this example indents lines by using spaces in the SHOW statements at 12070 and 19010. The SHOW statement has a print position option that could be used for such positioning.

THE RESULTING UPDATE SESSION

```
Ready

RUN TRQENH

TRANSACTION PROGRAM TRQENH

Enter purchase order number ? 10066
Vendor number? 11
Vendor # 11 not on VENDOR.INT
Do you wish to add to file <NO >? YES
Vendor name? THE LIGHTHOUSE
Street address <>? 765 MAPLE
City and state <>? GENEVA, IL
Zip code <>? 60105
     Order and vendor added to respective files
Enter purchase order number ?
     Processing complete. Have a nice day

END OF TRANSACTION PROGRAM TRQENH
```

EXAMPLE 3: Q SKELETON WITH VIDEO EFFECTS

Here is the same file definition with the addition of statements for screen formatting. (These added statements are printed here in bold for quick reference.) Explanation of the video statements follows the file definition.

```
11 !*******************************************************************
12 !TR SKELETON Q FILENAME: TRQV.DEF
13 !PURPOSE OF TR: UPDATE ORDER FILE; ADD NEW VENDOR INFO TO VENDOR FILE
14 !AUTHOR: S. HOPE
15 !DATE: MAY 13, 1982
16 !*******************************************************************
17 !
100   OPTIONS
110        SKELETON Q
120        TERMINAL VT100
200   DEFINE
210        YES.OR.NO S 3
300   FILES
310       FILE1 ORDER
320       FILE2 VENDOR
3500   LOGIC
5000       BEGIN PROGRAM
5010            CLEAR ALL
6000         Q10
6010            PROMPT PURCHASE.ORD.NO "Enter purchase order number " &
                EXIT " " REVERSE POS 5;1
6020            FIND FILE1 INDEX1 PURCHASE.ORD.NO ERROR=Q20 NOMESSAGE
6030            SHOW "Sorry, purchase order already in ORDER.INT":10;5
6040            GOTO Q10
7000         Q20
7010            ERROR% = 0% !clear FIND error from Q10 &

7020            !input vendor, but do not check vendor file &
                !(note STORE will skip external file check) &

7030            PROMPT F1.VENDOR.NO "Vendor number" STORE &
                CLEAR SCREEN REVERSE POS 5;5
7040       &
                 !look up vendor # on VENDOR.INT
7050            FIND FILE2 INDEX1 F1.VENDOR.NO ERROR=Q30 NOMESSAGE
7060            !vendor found on file; must be OK
7070            ADD FILE1
7080            !all done; repeat for next purchase order
7090            CLEAR ALL
7100            SHOW "Order entered":10;5
7110            GOTO Q10
8000         Q30
8010            ERROR% = 0%  !clear FIND error from Q20 &

8020            SHOW "Vendor #":10;7 F1.VENDOR.NO:18;7 " not on
VENDOR.INT":25;7
8030            PROMPT YES.OR.NO "Do you wish to add to file" DEFAULT "NO" &
                REVERSE POS 5;8
8040            IF YES.OR.NO <>"YES" AND &
                YES.OR.NO <>"NO"  THEN &
                SHOW "Enter   YES  or NO  please":5;10 &
                \GOTO Q30 &

8050            IF YES.OR.NO = "NO" THEN &
                GOTO Q20 &

8060            F2.VENDOR.NO = F1.VENDOR.NO  !assign vendor # &

9000         Q40
```

```
9010            PROMPT VENDOR.NAME "Vendor name" CLEAR LINE &
                REVERSE POS 5;10
10000        Q50
10010           PROMPT STREET.ADDRESS "Street address" REVERSE POS 5;12
11000        Q60
11010           PROMPT CITY.STATE "City and state" REVERSE POS 5;14
12000        Q70
12010            PROMPT ZIP.CODE "Zip code" REVERSE POS 5;16
12020           !add record to order file
12030           ADD FILE1
12040           !add record to vendor file
12050           ADD FILE2
12060           !Clear screen, report adds, return for next order
12070           CLEAR ALL
12080           SHOW "Order and vendor added to respective files":10;5
12090           GOTO Q10
19000       END PROGRAM
19010           CLEAR ALL
19020           SHOW "Processing complete. Have a nice day!":10;5
```

EXPLANATION OF VIDEO FILE DEFINITION

The third example has the same purpose as the second example. It differs only by the addition of video effects. Note that the OPTIONS section includes the statement TERMINAL VT100. Video statements include CLEAR ALL, CLEAR SCREEN, CLEAR LINE, REVERSE, and POS (or POSITION). The CLEAR statements clear the screen or line. REVERSE causes the video screen to be the opposite of the usual bright or dark setting. The space on the screen where the user's response will appear is thus highlighted.

Notice also that print positions have been added to the SHOW statements (in the format :x;y) to create helpful spacing. Every SHOW statement must have a print position specified. The print position statement :10;5 in line 12080 begins the message in the tenth horizontal position on vertical line 5.

DEMONSTRATION OF VIDEO EFFECTS

To see the effect of the video statements in this file definition, log into any Ross Systems timesharing account and give the command STUDY at the systems level. Necessary files will be copied to your account. Now give the command RUN STRQV and respond to the prompts as shown in the example below. (The printed page cannot duplicate the dynamic video effects. For example, the screen clearing between questions is not shown below. An effort has been made to duplicate the print position effects.) To erase the demostration files from your account, give the command RUN FINISH.

```
Ready

RUN TRQV

TRANSACTION PROGRAM TRQV

     Enter purchase order number ? 10066
     Vendor number? 11
          Vendor #11      not on VENDOR.INT
     Do you wish to add to file <NO>? YES
     Vendor name? THE LIGHTHOUSE
```

```
          Street address <>? 765 MAPLE
          City and state <>? GENEVA, IL
          Zip code <>? 60105
               Order and vendor added to respective files
          Enter purchase order number ?
               Processing complete. Have a nice day

     END OF TRANSACTION PROGRAM TRQENH
```

SKELETON Q OPTIONS SECTION

The OPTIONS sections is used to specify various processing options, such as the type of program to be generated (in this case, SKELETON Q) and the terminal type.

Each option has a keyword that is followed by one or more values indicating your choice. You may include options in the file definition in any order. The option need be specified only if the default is not appropriate. The OPTIONS section is not required if all the defaults are suitable.

EXAMPLE:

```
  100   OPTIONS
  110        SKELETON Q
  120        TERMINAL VT100
```

SUMMARY OF OPTIONS FOR SKELETON Q

KEYWORD	VALUES	DEFAULT	MEANING
CHAIN	filename	NONE	Name of a program to run after the TR program is completed (optional)
SEGMENT	filename	definition filename	Name to be given generated program (optional)
SKELETON	Q	Q	Type of program (optional)
TERMINAL	ASCII VT100	ASCII	Type of terminal
VALIDATE	ON OFF	ON	Allows suppression of validation rules in file definition. Validation will only occur for PROMPT items

DETAILED EXPLANATION OF OPTIONS

CHAIN filename

This keyword specifies the name of a program to be executed after the generated transaction program is completed. This program may be another INTAC program or any other type of program (BASIC, FORTRAN, PASCAL). Use this keyword to set up a stream of automatic processing.

EXAMPLE:

```
100   OPTIONS
200         CHAIN UPDATE
```

The program named UPDATE will be run after the transaction program has been completed.

DEFAULT: No chaining

SEGMENT filename1

The segment statement is optional for Skeleton Q because INTAC assumes that the Q Skeleton TR definition filename will be the name for the generated BASIC program. You may override this assumption by using the SEGMENT option to name the generated program. The name you assign may be any valid RSTS/E or VMS name, including account number (or VAX directory name), device specification, and protection code. INTAC will create a BASIC program with an extension of BAS (for example, QUEST1.BAS)

EXAMPLES:

```
100   OPTIONS
200         SEGMENT QUEST1

100   OPTIONS
200        SEGMENT [240,11]QUEST1
```

DEFAULT: SEGMENT filename1

SKELETON Q

Designates the structure of the program to be generated. Since the question and answer structure (Q) is the default, it is not necessary to include this keyword in the file definition.

TERMINAL ASCII or VT100

Specifies the type of terminal to be used. The default ASCII is appropriate unless you want to use screen formatting. To achieve screen formatting, you must have a VT100 compatible terminal and must specify VT100 in the OPTIONS section. In addition, you must include screen formatting detail in every LOGIC section SHOW statement.

A file definition that uses screen formatting will run on hardcopy terminals, but terminal display will be preceded by apparently meaningless characters.

EXAMPLE:

```
100   OPTIONS
200         TERMINAL VT100
```

DEFAULT: ASCII

VALIDATE ON or OFF

Specifies whether the answer will be validated automatically against the file definition edit parameters after it is prompted in the LOGIC section. (Data is always validated for data type.) Use VALIDATE OFF when you want to do your own data validation in the LOGIC section.

Similarly, the LOGIC section has statements for VALIDATE and STORE and the LOGIC section statement PROMPT makes available the options VALIDATE, STORE, and NOSTORE. VALIDATE and STORE share the store function. In addition to performing edit parameter checks, VALIDATE also maintains the answer to the prompt (a data item value) until the next prompt for the same item so that you can perform operations on that data item value.

If you want to by-pass the edit parameter checks, then specify VALIDATE OFF in the OPTIONS section. STORE becomes the default for PROMPT statements. The answer is not subjected to edit parameter checks, but it is available for your LOGIC section operations. If VALIDATE ON is specified in the OPTIONS section, (or assumed by default), then VALIDATE becomes the default for all prompt statements in the LOGIC section.

EXAMPLE:

```
100    OPTIONS
200          VALIDATE OFF
.
3500    LOGIC
7030              PROMPT F1.VENDOR.NO "Vendor number"
7050              FIND FILE2 INDEX1 F1.VENDOR.NO ERROR=Q30 NOMESSAGE
```

In this example, the PROMPT statement defaults to the STORE option because VALIDATE OFF has been specified in the OPTIONS section. The FIND statement causes INTAC to retrieve the record for the VENDOR.NO from FILE2 (VENDOR.INT). This use of VALIDATE OFF enables the TR author to have more control over the process of looking up the vendor in the VENDOR file. When no record is found on FILE2, the author can send the program on to prompt for additions to the VENDOR.INT file, turning off the usual INTAC error messages.

DEFAULT: ON

SKELETON Q DEFINE SECTION

The DEFINE section (in conjunction with the LOGIC section) allows you to define new data items. For example, define a new item calculated from items in more than one file. Or define a data item to be prompted for during an interactive update session. Use the defined items in the TR definition LOGIC section as though they were data items in one of the accessed files.

Each line in the DEFINE section defines one data item and is formatted with parameters similar to a data item in an INTAC file (see Chapter 3, CR).

This section is required only when defined items are to be used in a TR program.

FORMAT: `itemname type [xx.y] [DEFAULT "defltvalue"]`

where

`itemname`	consists of from 1 to 24 characters. The characters must be alphabetic, numeric, or periods. The first character must be alphabetic. No spaces or other special characters allowed. Names which are INTAC or BASIC reserved words may not be used (See Appendix D for a list of reserved words).
`type`	data type, as used in INTAC files: S character string R real (floating point) number I integer D date field
`xx.y`	print format xx is the total width of printed field including commas, decimal points and sign or parentheses. For a string field this is the length of the string. For a DATE field, this will be either 8 or 10. y is the number of decimal places in floating point numbers only. A limit of sixteen digits exists for REAL variables.
`DEFAULT "defltvalue"`	This optional value will be used as the default for this item. If not specified, no default value will be assumed.

DEFINE SECTION EXAMPLE:

```
200   DEFINE
210          YES.OR.NO S 3
```

WHEN TO DEFINE ITEMS

This chapter suggests that you plan the LOGIC section before the DEFINE section so that you will know exactly which new data items you need to DEFINE. As you write the LOGIC section, you can use any valid BASIC variables. (These "variables" may be considered new, temporary data items.)

Do not DEFINE an item used as a control variable in the LOGIC section (e.g. Do not DEFINE to control a loop). For best results, use a BASIC variable (e.g. ICOUNT) when control variables are needed in the LOGIC section.

A maximum of 99 items may be defined in any INTAC TR definition file.

SKELETON Q FILES SECTION

The FILES section specifies the INTAC files to be used. Any files externally referenced by an INTAC file definition must also be specified in the FILES section.

FILES SECTION STATEMENTS

```
FILEn filename [READONLY]
```

This statement specifies a file to be used by the generated program. Up to three INTAC files may be specified, for n=1, 2 and 3. (FILE3 cannot be specified if there are one or more external statements specified. See below, EXTERNAL filename.)

If you will use an INTAC file only to validate or retrieve information (not to update), READONLY may be specified. This option protects the file by preventing the user from adding, updating, or deleting records from the file. It is a useful option when multiple users are allowed.

```
EXTERNAL filename
```

An EXTERNAL filename statement should be used when an external file reference occurs in the INTAC file definition of a file to be accessed. Items from a file listed only with the EXTERNAL statement cannot be referenced. If you want to reference items from a file, then specify it with a FILE2 statement. (It is not necessary then to specify the file with an EXTERNAL statement.) There is no limit to the number of EXTERNAL statements in a TR program. The file specified as FILE1 cannot be specified as an EXTERNAL file.

FILES SECTION EXAMPLE:

```
300    FILES
310        FILE1 ORDER
320        FILE2 ASSET READONLY
330        EXTERNAL VENDOR
```

In this example, the INTAC files ORDER and ASSET are being used. The EXTERNAL file VENDOR is being used by another file (ORDER) as a check to ensure the values entered for an item (VENDOR.NO) exist in the validation file (VENDOR). This external reference file must be specified separately. Note that ASSET is being used only to retrieve information and is specified as READONLY.

SKELETON Q LOGIC SECTION

The LOGIC section specifies the processing rules for the various parts of the program. The LOGIC section allows the full flexibility of BASIC statements as well as INTAC statements. Note that LOGIC statements allow interrogation of the data in the file before updating operations are performed.

The Skeleton Q LOGIC section consists of subsections labeled Qn from Q1 up to Q32767. Control of the program can be transferred from subsection to subsection based upon the answers to prompts or interrogation of the data in the file.

In the definition file, lines 3500 through 19999 are reserved for the LOGIC section.

SUMMARY OF SKELETON Q LOGIC STATEMENTS

In addition to the INTAC LOGIC section statements listed below, you may include your own BASIC code.

STATEMENT	PURPOSE
ADD	Adds a record to an INTAC file
CLEAR	Clears the line or screen before prompting (VT100 only)
DELETE	Deletes the current record on an INTAC file
EXIT	Transfers control of the TR program to the END PROGRAM section
FIND	Searches for and retrieves a record with a specific value
FINDNEXT	Retrieves the next record
GOTO	Transfers control of the TR program to a specified question number
POSITION	Moves the VT100 cursor to the specified position on the screen
PROMPT	Defines prompting; has several options, including video effects
SHOW	Specifies data items to be displayed and position on screen or page
STORE	Temporarily stores the answer just entered by user (used instead of VALIDATE command)
TEST	Tests the answer to a prompt and transfers control of the program based upon the test

UPDATE	Updates the current record
UPDATI	Changes the index for the current data record
VALIDATE	Validates and stores the last answer given

The statements CLEAR and POSITION as well as the PROMPT options REVERSE and UNDERSCORE, allow screen formatting on VT100 compatible terminals. To use these capabilities, you include the keyword TERMINAL VT100 in the OPTIONS section. Your TR program can run on a hardcopy terminal; however, apparently meaningless characters will be printed before the terminal display.

LOGIC SUBSECTIONS

Each question to be asked is defined in a LOGIC section subsection labeled Qn. Subsections labeled BEGIN PROGRAM and END PROGRAM can also be used. The TR facility will transfer from one subsection to the next automatically or you may control transfer to another question number by using the GOTO statement. Each question subsection is normally used to prompt for and process one INTAC data item. If you use a GOTO statement and specify a line number, it should be within the same question subsection or in the CODE section.

There may be no more than 100 questions. No question may be give a number greater than Q32767. Question labels may be in any order in the definition file, but will always be processed in ascending numerical order (e.g. Q10, Q20, Q30).

```
BEGIN PROGRAM
    :
    :
    Q10
    :
    :
    Q20
    :
    :
    Q30
    :
    :
              IF...THEN...Q20
    :
    :
END PROGRAM
```

Between the end of the statements for one question and the beginning of statements for the next, there must be at least one available line number. Each Qn subsection may contain only one PROMPT statement. A PROMPT statement may not exist outside of a Q subsection.

DETAILED DESCRIPTIONS OF LOGIC STATEMENTS

The following LOGIC statements are described in alphabetical order for easy reference. In addition to these INTAC statements, you may use BASIC statements. For example, you may assign a value to an INTAC or DEFINE item (e.g. DEPT.NO=NEW.DEPT.NO).

ADD FILEn [ERROR = Qn] [NOMESSAGE]
[ERROR = lineno]

Adds a record and indexes to INTAC file n as defined in the FILE section. This statement assumes that the processing in some way has filled the record with values for all data items (other than DATE LAST EDIT and DELETE FLAG). If any data items have not been given values, random characters may appear in the updated file.

If an error occurs, control is transferred to the question or line number specified in the ERROR option. If you specify a line number, it should be within the same question subsection or in the CODE section. An error message is printed first unless NOMESSAGE is specified. If you omit ERROR=, the program exits on any error.

Use the NOMESSAGE option when you want to create your own error-handling procedure. See Appendix B for error-handling information.

EXAMPLE: `ADD FILE1 ERROR = Q20`

CLEAR [ALL]
[LINE]
[SCREEN]

For VT100 compatible terminals, you can use this statement to clear the entire screen, clear to the end of the current line, or to the end of the screen. Use TERMINAL VT100 in the OPTIONS section. The CLEAR statement will be ignored if you omit the TERMINAL VT100 option in order to run the TR program on a hardcopy terminal.

EXAMPLES:

```
CLEAR LINE
CLEAR SCREEN
```

DELETE FILEn [ERROR = Qn] [NOMESSAGE]
[ERROR = lineno]

The DELETE statement will delete the current record and indexes on INTAC FILEn. If an error occurs, control is transferred to the question or line number specified in the ERROR= option. If you specify a line number, it should be within the same question subsection or in the CODE section. An error message is printed first unless NOMESSAGE is specified. If the ERROR = option is omitted, the program exits on any error. See Appendix B for error-handling information.

EXAMPLE: `DELETE FILE2 ERROR = Q20`

EXIT

The EXIT statement causes control to be transferred to the END PROGRAM section or terminates the program if there is no END PROGRAM section.

EXAMPLE: `EXIT`

FIND FILEn INDEXn value [ERROR = Qn] [NOMESSAGE] [ERROR = lineno]

The FIND statement retrieves a record in the specified file using the value given for the specified index. FILEn is the file as specified in the FILE section. INDEXn is the index number in the file definition.

Value can be an INTAC data item name, a DEFINE data item name, a BASIC variable, or an actual value for a data item or variable or finally, a combination of the above joined with a plus sign (+).

If an error occurs, control is transferred to the question or line number in the ERROR= option. If you specify a line number, it should be within the same question subsection or in the CODE section. An error message is printed first unless NOMESSAGE is specified. If the ERROR = option is omitted, the program exits on any error.

If you specify INDEX0, the record will be retrieved by record number. When you retrieve by record number, be sure that whatever you are using for value will be an integer. FIND by record number will transfer control to an ERROR routine (if any is specified) when it finds a deleted record. The value of the error flag (ERROR%) will be -7. See Appendix B for information on error handling using the FIND command.

EXAMPLES:

```
FIND FILE1 INDEX1 10015 ERROR = Q20
```

This first example will get a record from FILE1 (which in the chapter example is ORDER.INT) with INDEX1 (in the ORDER file, the index on PURCHASE.ORD.NO) equal to 10015. (In this example, value is a data item value.) Control will transfer to question 20 if there is an error.

```
FIND FILE1 INDEX1 PURCHASE.ORD.NO ERROR = Q20
```

This second example will get a record from FILE1 with INDEX1 equal to the value in the data item PURCHASE.ORD.NO. When an error condition is detected, processing will return to question 20.

FINDNEXT FILEn INDEXn [value] [ERROR = Qn] [NOMESSAGE] [ERROR = lineno]

The FINDNEXT statement gets the next record from INTAC FILEn using INDEXn. FINDNEXT must be preceded by a FIND on the same index. FINDNEXT cannot be used for INDEX0(record number). If an error occurs, control is transferred to the question or line number specified in the ERROR= option. An error message is printed first unless NOMESSAGE is specified. If the ERROR = option is omitted, the program exits on any error.

EXAMPLE: `FINDNEXT FILE1 INDEX3 ERROR = Q20`

GOTO Qn

This statement transfers control to the specified question or line number. If you specify a line number, it should be within the same question subsection or in the CODE section.

POSITION x;y

For VT100 compatible terminals, use the POSITION statement to position the cursor to the specified line y and column x. Use TERMINAL VT100 in the OPTIONS section.

EXAMPLE: `POSITION 3;7`

This example will position the cursor to line seven (7) and horizontal position three on the line. If necessary, use a formatting ruler to estimate the horizontal position.

PROMPT item ["custom message for prompt"] [option]

This statement defines the question to be asked when the TR program is run. Use a separate Skeleton Q Qn subsection for each PROMPT statement. A PROMPT statement may not exist outside of a Q subsection. The shortest form of the statement is: `PROMPT item`

Item may be an INTAC data itemname or an itemname from the DEFINE section. If you use this short form of the statement, TR will use the itemname as the prompt. Item may be followed by "custom message for prompt " which is your own prompt message to be used in place of the itemname at the time of execution of the program.

PROMPT OPTIONS:

The following options may be added to the PROMPT command. They may be given in any order and should be separated from each other by a space.

BACKSLASH Qn
CLEAR LINE/SCREEN/ALL
DEFAULT "value"/NODEFAULT
EXIT "value"/NOEXIT
HELP "text"
POSITION x;y
REVERSE/UNDERSCORE
VALIDATE/STORE/NOSTORE

These options are described in detail below in alphabetical order.

BACKSLASH Qn

Specifies the question to which control is to return if a backslash (\) is entered for this question. If you do not include this option, control returns to the previous question.

CLEAR ALL
CLEAR LINE
CLEAR SCREEN

For a VT100 compatible terminal, you may use this option to clear the entire screen, clear to the end of the line or to the end of the screen before prompting. Use TERMINAL VT100 in the OPTIONS section.

DEFAULT "value"
NODEFAULT

Specifies a default value to be used in place of any default listed in the file definition. Note that the value must be enclosed in quotation marks. NODEFAULT suppresses use of the file default value and forces the user to enter a value. If DEFAULT is specified or the file default is used, the default value will be displayed in the prompt. If you omit this option, the file definition default value (if any) remains in force.

HELP "text"

Specifies a help message to be displayed when the user responds to the prompt message by entering ? or HELP.

POSITION x;y

For VT100 compatible terminals, this PROMPT keyword positions the prompt message at the VT100 screen position column x, line y (e.g. POSITION 1;5 means put the cursor in position 1 and line five (5) of the VT100 terminal screen). You may abbreviate the PROMPT keyword POSITION as POS. Use TERMINAL VT100 in the OPTIONS section.

REVERSE
UNDERSCORE

For VT100 compatible terminals, this option causes the space for the prompt answer to be displayed in reverse video or as an underscore. Use TERMINAL VT100 in the OPTIONS section.

VALIDATE
STORE
NOSTORE

This option indicates whether you want INTAC to validate and store the answer automatically after prompting or not. Validation is done using the edit parameters in the file definition for this item. VALIDATE validates and stores the answer in the item. STORE stores the answer temporarily without validating. NOSTORE indicates that the answer will be neither validated nor stored. If you use STORE, you will normally include statements to do your own validating. If you use NOSTORE, you will normally be doing your own testing and validating.

If you omit these options, the default depends upon your specification in the OPTIONS section of the TR definition file.

PROMPT EXAMPLES:

```
PROMPT VENDOR.NO "Vendor Number" VALIDATE
```

In this first example the INTAC item VENDOR.NO will be prompted for with the message "Vendor Number?". If there is a default value in the file the message will be "Vendor Number <default>?". The value entered will be validated using the MINIMUM, MAXIMUM, TABLE, or EXTERNAL FILE specified in the file definition for this item. If the entered value is not valid, the prompt will be repeated. If it is valid, the entered value will be stored in the item.

```
6520 PROMPT VENDOR.NO POS 1;5 CLEAR SCREEN UNDERSCORE NOSTORE
6530 TEST = "UNKNOWN" THEN Q17
6540 VALIDATE
```

In this second example the INTAC item VENDOR.NO is prompted for with the message "VENDOR.NO?" A file default would appear as "VENDOR.NO <default>?" The screen is cleared before the prompt message is printed. The prompt message will begin at line 5, column 1 on the screen. The space for the answer will be underscored. The NOSTORE statement in line 6520 enables the TR author to use the TEST statement in line 6530 to take appropriate action if the user enters the word "UNKNOWN". After this test, data is validated and stored as a result of the VALIDATE statement in line 6540.

SHOW element1 element2 ... elementn [+]

The SHOW statement is used to specify the data elements to display. The elements may be INTAC data items, DEFINED items, BASIC variables or a text message enclosed in quotes. The SHOW statement may also specify a print position across the screen or page and a format. When using video effects for VT100 terminals, you must specify print position in every SHOW statement.

If a SHOW statement ends with a plus sign (+), the print line is to be continued on another SHOW statement.

For file definitions in which no special video effects are used, each element in the SHOW statement may take the form:

```
itemname[:positionn][,format]
"text message"[:positionn][,format]
```

For file definitions using video effects and specifying TERMINAL VT100 in the OPTIONS section, each element in the SHOW statement can take the form:

```
itemname[:positionn;z][,format]
"text message"[:positionn;z][,format]
```

where

`itemname`	any INTAC data item name, DEFINE data item name, or BASIC variable or constant
`"text message"`	Any text message
`(nonvideo) positionn`	For hardcopy terminals, a specific print position on the line. positionn may be a number (from 1 to the maximum width of your terminal) or an expression that can be evaluated at execution time. If a position is omitted, the item will print in the next available print position. If the VT100 format is being used, refer to the special position statement (below) for cursor movement.
`(VT100 format) positionn;z`	positionn;z is a specific print position on the screen. positionn is a number from 1 to the maximum width of your terminal-- usually 132) indicating the character position across the screen. z is a number from 1 to 24 indicating the line on the screen. The print position is required if the VT100 option is specified in the OPTIONS section.
`format`	xx.y is an INTAC print format. If it is omitted, the format is taken from the item definition. If the position is omitted and the format is included, the format must be preceded by a comma. xx is the width of printed field including commas, decimal point and sign. y is the number of decimal places.

SHOW EXAMPLES:

```
SHOW VENDOR.NO VENDOR.NAME TOT.AMT
```

In this first example, each of the data items will be printed in the next available print position. All of the elements except for TOT.AMT are data items in the INTAC file and INTAC will print them from the file. TOT.AMT was defined as a new data item and used in the LOGIC section to hold the results of a calculation. The calculated results will print.

```
SHOW PURCHASE.ORD.NO:1 DESCRIPTION:10 ASSET.TYPE:33,2.0 +
SHOW VENDOR.NO:40 LIFE:48,10.2
```

In the above example, the first data element is to be printed in position 1. The second element (DESCRIPTION) will be printed in position 10 on the line using its INTAC file default print format. The third element, ASSET.TYPE, will be printed in position 33 on the line, using a format of 2.0. The two fields defined in the second SHOW statement will print on the same line in print positions 40 and 48, respectively.

```
SHOW "Vendor #" F1.VENDOR.NO:12 "  not on VENDOR.INT"
```

This example will print the text beginning at print position one and the INTAC item F1.VENDOR.NO beginning at position twelve followed in the next available print position by the text.

STORE

This statement temporarily stores the last answer to the last prompt. A check for data type is performed, but not a check against edit parameters. (STORE performs the same store function as does the VALIDATE statement. However, VALIDATE does more: it validates as well as temporarily stores the last answer to the last prompt. Validation is done using the edit parameters in the file definition for this item.)

TEST operator "expression" THEN lineno
Qn

The TEST statement tests the user's answer to the last PROMPT statement and goes to the specified line or question number if the test is true. Operator must be a comparison operator (>, <, >=, <=, =, <>, ==) and "expression" must be a string value (that is, a text). The user's answer is compared with "expression", a string.

EXAMPLE:

```
PROMPT ACTION "ENTER UPDATE, DELETE, OR STOP"
TEST = "STOP" THEN 32000
TEST = "UPDATE" THEN 33000
TEST = "DELETE" THEN Q3
```

In this example, the value entered is contained in the DEFINED item ACTION and is compared to the texts STOP, UPDATE, or DELETE.

UPDATE FILEn [ERROR=lineno] [NOMESSAGE]
[ERROR=Qn]

The UPDATE statement will change the current record on INTAC FILEn. If an error occurs, control is transferred to the line or question number specified. An error message is printed first unless NOMESSAGE is specified. If you specify a line number, it should be within the same question subsection or in the CODE section. If the ERROR = option is omitted, the program exits on any error. UPDATI should be used after UPDATE if an index value has changed.

TR

EXAMPLE: `UPDATE FILE2 ERROR = Q40`

UPDATI FILEn INDEXn old.value [ERROR =Qn] [NOMESSAGE]
[ERROR=lineno]

The UPDATI statement changes the specified index for the current data record from the old value given to the new value formed from the current value of the items in the record. Used after FIND and UPDATE if an index item value changes. If you specify a line number in the ERROR= option, it should be within the same question subsection or in the CODE section.

EXAMPLE: `UPDATI FILE1 INDEX3 VALUE ERROR = Q70`

VALIDATE

The VALIDATE statement validates and stores the last answer to the last prompt. Validation is done using the edit parameters in the file definition for this item. This statement is typically used within a question sequence following a NOSTORE or a PROMPT and in conjunction with specification of VALIDATE OFF in the OPTIONS section.

EXAMPLE: `VALIDATE`

SKELETON Q CODE SECTION

The CODE section contains user subroutines and functions. Lines 20000 through 23999 are reserved for the CODE section. The line numbers specified in the CODE section (unlike the line numbers in other sections) are not changed by the transaction program generator.

The CODE section may contain any BASIC or TR LOGIC section statements and variables, except the PROMPT statement.

The CODE section may only be referenced from the LOGIC section. Control is transferred to the CODE section from the LOGIC SECTION in one of the following ways:

- function call in a BASIC statement
- GOTO or GOSUB in a BASIC statement
- ERROR = line number in a LOGIC statement.

If the CODE section is used, it must be the last section in the report definition file. See Appendix B for error handling information.

SKELETON B: SORT AND BREAK

Use Skeleton B for file updating requiring sorting; for example, to update records selectively throughout one or more files based upon your criteria. Include criteria directly in the SELECT section of the TR file definition or create custom prompting to request information at run time to establish the criteria for record selection and processing.

The system you set up with Skeleton B can control the entire process so that an inexperienced person can run the update.

HOW TR SKELETON B WORKS

In order to follow your selection criteria, TR creates a program to sort records. It creates a second program or program segment to perform specified operations at points where certain values (such as department number) change. The points in the processing at which these values change are called "control breaks" and the skeleton itself is often called the "sort and break" skeleton.

In the SELECT section of the Skeleton B definition file, you define the data and files for the first program to select and sort. The second program segment uses your specifications in the LOGIC section to process the selected and sorted data.

SUMMARY OF SKELETON B SECTIONS

A file definition using SKELETON B may include sections as shown on the following chart. The section name is on a line by itself and is followed by information appropriate to that section. Line numbers must keep the sections in the order shown on the chart below.

SECTION	DESCRIPTION	PAGE
OPTIONS	Specifies various processing options, such as data validation, and terminal type (for cursor control). Specify here that SKELETON B should be used. The names of the two generated BASIC programs must also be specified here.	11-39
DEFINE	Defines temporary data item used by the transaction program. These are data items are not found in any of the INTAC files. (This section is optional.)	11-42
FILES	Specifies the INTAC files to be used in the generated program. Files referenced by external file specifications must also be listed in this section.	11-44

SELECT	Indicates the records to be selected, the sort sequence, and control break fields (see explanation of breaks in this chapter).	11-45
LOGIC	Specifies the processing rules for the sorted records. The format of this section follows the pattern of control break fields established in the SELECT section.	11-49
CODE	Reserved for user BASIC subroutines and functions used for processing the data. (This section is optional.)	11-60

PLANNING THE B SKELETON TR

As you write a B skeleton transaction definition file, we recommend that you write sections in an order that reflects your planning process: FILES, SELECT, LOGIC, CODE, DEFINE, OPTIONS. Assign line numbers to keep the file in the order necessary for the TR facility.

With the B Skeleton, the data in a file drives the program. Decide which file should drive it and make that FILE1 in the FILES section. Then plan the SELECT section with appropriate INCLUDE or EXCLUDE statements and control breaks. In the LOGIC section, put in the BEGIN and END statements based on the breaks in the SELECT section and fill in necessary LOGIC.

The data items you will need in the DEFINE section will be clear to you after you complete the SELECT and LOGIC sections. Last fill in the OPTIONS section, using SKELETON B and giving names to the two generated program segments.

THE CONCEPT OF CONTROL BREAKS

Essential to an understanding of sort and break programs is the concept of the control break. The purpose of a control break is to allow specified operations or events to occur at appropriate moments as data is processed. You define control breaks with a BREAK statement in the SELECT section of the definition file. Organize the LOGIC section into subsections based upon these control breaks. (Detailed instructions are given in the parts of this chapter that describe the SELECT and LOGIC sections.)

A control break will occur when the value of an item specified in the BREAK statement changes. For example, if a program is to create detail records for each employee and print a subtotal by department, then a control break will occur when the department number read for a record is different from that of the previous record.

TYPICAL CONTROL BREAK EVENTS

You use control breaks to cause events such as the following:

- keep a count of records for a group
- At the beginning of a new group defined by a control break, print group headings

or clear the screen (TERMINAL VT100 option).

- At the end of a group, print group totals and clear the total accumulators for the next group.

You can see that programs may involve processing at different points:

- at the beginning of the various control breaks
- the detail processing of a data record
- at the end of the control breaks.

CONCEPTUAL DESCRIPTION OF SORT/BREAK PROGRAMS

We can describe the processing that occurs in a typical sort and break program as follows:

```
100 BEGIN PROGRAM          !BREAK0            (initialize program)
220 BEGIN DIVISION         !BREAK1            (Print DIVISION headings)
320 BEGIN DEPARTMENT       !BREAK2            (Print          DEPARTMENT
                                              headings)
420 DETAIL PROCESSING                         (Prompt    for    NAME    and
                                              AMOUNT;  Update  AMOUNT  on
                                              file;           Accumulate
                                              DEPARTMENT,  DIVISION,  and
                                              PROGRAM totals)
520 END DEPARTMENT         !BREAK2            (show DEPARTMENT totals)
720 END DIVISION           !BREAK1            (show DIVISION totals)
1000 END PROGRAM           !BREAK0            (show REPORT totals)
```

TWO EXAMPLES OF SKELETON B

Two B Skeleton examples are included in this chapter. The first simple example changes one data item throughout the ASSET file and keeps a count of the number of changes for each department. The second example illustrates a common situation: the company has been reorganized to combine some departments. The assets must be assigned to these new departments.

For each of these examples, the chapter includes the file definition, an explanation, and a sample update session.

SIMPLE SKELETON B EXAMPLE

Note the organization of the LOGIC section by control breaks. This example uses suggested standard line numbers and illustrates the use of the ampersand to continue lines. This file definition was created using the ROSS EDITOR.

```
11 !***********************************************************
12 !TR SKELETON B FILENAME: TRBSIM.DEF
13 !PURPOSE OF TR: CHANGE ASSET TYPES AND LIFE IN ASSET.INT
14 !AUTHOR: S. HOPE
15 !DATE: MAY 15, 1982
16 !***********************************************************
17 !
100   OPTIONS
```

```
110          SKELETON B
120          SEGMENT TRBSIM TRBSM2
200     DEFINE
210          CHG.TYPE S 1 PROMPT "Enter Asset type to change"
220          NEW.LIFE I 3 PROMPT "New Asset life (in years)"
230          COUNT I 3
300     FILES
310        FILE1 ASSET
340     SELECT
350          GET1
360          INCLUDE IF ASSET.TYPE = CHG.TYPE
370          SORT DEPT.NO
380          BREAK DEPT.NO
3500     LOGIC
5000         BEGIN PROGRAM
6000         BEGIN DEPT.NO
6010              COUNT = 0          !reset count for next dept
12000        DETAIL PROCESSING
12010              LIFE = NEW.LIFE           !assign new asset life &
                                             !then update file &

12020              UPDATE FILE1
12030                                        !increment dept count
12040              COUNT = COUNT + 1
18000        END DEPT.NO
18010             SHOW "Department #" DEPT.NO ":" COUNT:25,3 " asset lives
changed"
19000        END PROGRAM
```

EXPLANATION OF SIMPLE B SKELETON TR

The purpose of this TR program is to change values for one data item throughout the case study ASSET.INT file. Four possible asset types are listed in the file definition: F (office furniture), C (computer equipment), E (miscellaneous), and O (other). The Tri-City planner wants all of the assets with a type O to have a life of 5 years. This TR retrieves all the O type assets and changes the associated LIFE values to 5. The TR is made more flexible by prompting for values for asset type and for LIFE so that it can be used again for similar changes.

The OPTIONS section specifies the B Skeleton. It names the two necessary TR programs in line 120.

The DEFINE section sets up two new data items (CHG.TYPE and NEW.LIFE) to receive values through prompting at run time. The new data item CHG.TYPE will hold the value of the asset type to be changed. It will be used in both the SELECT section and the LOGIC section. NEW.LIFE will be used only in the LOGIC section. It is a good practice to prompt in the DEFINE section rather than in the LOGIC section whenever possible. In this case, prompting must be done in the DEFINE section because the value for CHG.TYPE is needed in the SELECT section. A BASIC variable COUNT is included in the DEFINE section.

The FILES section lists the only file involved in this program, ASSET.INT as the primary file. In the SELECT section, the GET1 statement reads every record in this primary file. In line 360, the INCLUDE statement causes the program to select only those records that have an asset type matching the value entered by the user in response to the DEFINE section prompt. Lines 370 and 380 sort these records by department number so that a count of changes per department can easily be maintained. The BREAK statement indicates that processing will take place whenever the department number changes (in this case, a message is printed and the count will be reset). The program created by this SELECT section will select and sort the records before they are processed by the program defined by the LOGIC section. Only the records selected will be acted upon by the LOGIC section.

The LOGIC section sets up the necessary processing. At line 6000, the BEGIN statement indicates the data item break. Line 6010 defines a BASIC variable named COUNT that will keep track of the number of changes made for each department. This variable will be reset to zero whenever the department number changes. Notice that it is not necessary to include this BASIC variable in the DEFINE section because it is not used for prompting, nor is it used in the SELECT section.

For every record, the DETAIL subsection line 12010 changes the value of the data item LIFE to the value entered by the user in response to the DEFINE section prompt in line 220. Line 12020 updates the file for each record and line 12040 increases the count by one. As the program is run, line 18010 causes the display of a message reporting the number of records changed per department. The data item COUNT is positioned to begin in print position 25. It is given a print format of 3 so that the count can be as high as 999.

THE RESULTING UPDATING SESSION

When the program is run, the dialogue appears as follows.

```
RUN TRBSIM

TRANSACTION PROGRAM TRBSIM

Enter Asset type to change? 0
New Asset life (in years)? 5
4 RECORDS SELECTED
Department #  8501:       3 asset lives changed
Department #  9001:       1 asset lives changed

END OF TRANSACTION PROGRAM TRBSM2
```

SECOND B SKELETON TR EXAMPLE

```
11 !*****************************************************************
12 !TR SKELETON B FILE NAME: REORG.DEF
13 !PURPOSE OF TR: Change Dept #'s in Asset file after re-organization
14 !AUTHOR: S. Hope
15 !DATE: 15-June-1982
16 !*****************************************************************
17 !
100   OPTIONS
110        SKELETON B
120        SEGMENT REORG REORG2
200   DEFINE
210       NEW.DEPT.NO I 6
220       COUNT I 3
300   FILES
310       FILE1 ASSET
320       FILE2 DEPT
340   SELECT
350        GET1
380        SORT DEPT.NO
390        BREAK DEPT.NO
3500   LOGIC
```

```
5000       BEGIN PROGRAM
5010          SHOW "Department re-organization update"
5020          SHOW "  For each old department, enter the corresponding"
5030          SHOW "  new dept #. If the same dept is entered, no changes"
5040          SHOW "  will be made to the Asset file for that department."
5050          SHOW "  "
6000       BEGIN DEPT.NO
6010          SHOW "Old department: " DEPT.NO
6020          PROMPT NEW.DEPT.NO "Enter new department #"
6030          IF NEW.DEPT.NO = DEPT.NO THEN &
                    CHG.FL% = 0% &
              ELSE CHG.FL% = -1% &

6040          GOSUB 20100           !check if valid dept &

6050          IF BAD.DEPT% = -1% THEN &
                  GOTO 6020         !try again &

6060                                !print a blank line &
                                    !then dept info &
              SHOW " " &
              \SHOW "Dept name: " DEPT.NAME:15 &
              \SHOW "Manager:" MANAGER:15 &
              \COUNT = 0 &

12000      DETAIL PROCESSING
12010         IF CHG.FL% = -1% THEN &
                 DEPT.NO = NEW.DEPT.NO!change dept # &
                 \COUNT = COUNT + 1 !increment count &
                                        !update file &
                 \UPDATE FILE1 &

12020              ! NOTE that update must be on last sub-line &

18000      END DEPT.NO
18010            IF CHG.FL% = -1% THEN &
                    SHOW COUNT:10,5 " asset records updated" &

18020            IF CHG.FL% = 0% THEN &
                    SHOW "..... no changes made for dept" &

18030            SHOW " " &
                 \SHOW " " &

19000      END PROGRAM
20000    CODE
20100    !*****************************************************************&
         !*** Subroutine to check if DEPT.NO exists on DEPT file. &
         !*** If not on DEPT, then it is an invalid DEPT.NO. &
         !*****************************************************************
20110         FIND FILE2 INDEX1 NEW.DEPT.NO ERROR=20190 NOMESSAGE
20120         BAD.DEPT% = 0%            !clear bad dept flag &
              \RETURN               !no error, dept found &

20190         SHOW "Dept #" NEW.DEPT.NO " not in department file (DEPT.INT)" &
              \SHOW "...please try again" &
              \BAD.DEPT% = -1%      !set bad dept flag &
              \ERROR% = 0%          !clear error flag &
              \RETURN &
```

EXPLANATION OF SECOND B SKELETON TR

The purpose of this TR program is to adjust the ASSET file department numbers to reflect a reorganization of the Tri-City Company. Because of the reorganization, some assets must be assigned new department numbers. All of the assets for any given department will require exactly the same change, so if they are sorted by department, the changes for any department can be made efficiently. This program sorts the asset records by department and breaks on department number to allow prompting and other processing.

The changes in department number have already been made in the DEPT.INT file. This TR program will create custom validation of the new department numbers against the numbers in the DEPT.INT file. (The ASSET.INT file definition does not specify any kind of validation for department numbers.) The TR will also keep a count of the number of changes per department number.

At run time, this program will provide a brief explanation to the user and then, department by department, (not asset by asset) the user will be prompted for a new department number. Validation of the department number will be done against the DEPT.INT file, changes will be made to records, and the user will be given an appropriate report of update and of the number of records changed. If the department number entered by the user matches the record, the program will report that no changes have been made.

200 DEFINE
The DEFINE section includes a new data item named NEW.DEPT.NO to contain the response to a prompt in the LOGIC section. Prompting must occur in the LOGIC section for this program because the user must indicate the appropriate new department number for each old department.

300 FILES
The FILES section lists the ASSET file as the primary file and DEPT as the secondary file. Since this program uses DEPT.INT only for custom validation (no changes will be made to DEPT.INT), it could be specified as a READONLY file.

340 SELECT
The SELECT section creates a program to read every record, sort the records by department number, and break on department number for prompting and processing to be specified in the LOGIC section. In this program, all records in ASSET.INT will be selected for processing in the LOGIC section.

3500 LOGIC
The LOGIC section uses the BEGIN PROGRAM break to instruct the user. The instructions will be printed only once as the session begins. Line 6000 begins the processing by department number. AT line 6010, the SHOW statement causes display of the text message and the DEPT.NO of the first sequential record for a department. Line 6020 prompts for a value for the new department number.

Line 6030 (with its sublines) contains a BASIC IF-THEN-ELSE statement to test if the prompted department number matches the data item value on the ASSET file. A BASIC variable named by the author CHG.FL% is assigned the value 0% if the new department number matches and the value -1% if there is no match. (This variable need not be included in the DEFINE section. The % sign means that the value should be stored as an integer to save storage space in the computer.) This variable will be used in lines 12010 and 18010.

Next, the program checks to see if the department number entered by the user exists on the DEPT file. The author has put this validation in a subroutine in the CODE section. Line 6040 sends the program to this subroutine. The FIND statement at line 20110 retrieves the department number from secondary file DEPT. If no record is found, INTAC considers it an error and sends the program to line 20190. The NOMESSAGE option suppresses the usual INTAC error messages because line 20190 creates a custom message. This line also assigns a value (-1%) to an error flag (named by the author BAD.DEPT%). The value of this BAD.DEPT flag will direct the program appropriately.

The last subline of line 20190 returns control to line 6050 and then to line 6020 to prompt for department number again.

If the validation routine finds the department number on the DEPT file, then line 20120 ensures that the bad department flag is reset to 0% (meaning the department number is OK) and then returns control to line 6050 which will continue the program to line 6060. SHOW statements display the department name beginning in the horizontal position 15 across the page and the manager on the next line, also at the fifteenth print position. The last subline assigns the value zero to a BASIC variable to contain the count of records changed for the department.

Detail processing begins at line 12010. The BASIC variable CHG.FL% received a value of -1% in line 6030 when the department number entered by the user did not match the department number on the ASSET file record. Whenever the CHG.FL% has the value -1%, line 12010 will change the record department number to the new number entered by the user. The count variable will be increased by 1. The statement UPDATE FILE1 updates the primary file, ASSET.INT. Notice that line 12010 contains a BASIC assignment statement and an INTAC UPDATE statement. It is alright to have multiple statements on one line number. The INTAC file statement must be the last statement on the line number.

At line 18010, processing to occur at the change of department numbers is indicated. The count is reported. Line 18030 creates two blank lines to provide spacing between departments. The END PROGRAM statement is optional since no processing occurs at the end of this program.

THE RESULTING UPDATING SESSION

Below is a part of the updating session that occurs when this TR program is run. All departments in the file are processed by the program, but below are shown only enough departments to show how the program handles different situations.

```
RUN REORG

TRANSACTION PROGRAM REORG

77 RECORDS SELECTED
Department re-organization update.
  For each old department, enter the corresponding
  new dept #. If the same dept is entered, no changes
  will be made to the Asset file for that department
```

```
Old department:   1001
Enter new department #? 1501

Dept name:     FINANCE


Manager:       PETERS
             13 asset records updated

Old department: 2001
Enter new department #? 1501

Dept name:     FINANCE
Manager:       PETERS
              6 asset records updated

Old department:    3001
Enter new department #? 3001

Dept name:     PRESIDENT
Manager:       EWING
..... no changes made for dept

Old department:    7001
Enter new department #? 7001
Dept #  7001 not in department file (DEPT.INT)
...please try again
Enter new department #? 7501

Dept name:     COM. PROD
Manager:       SCUDAMORE
              3 asset records updated


END OF TRANSACTION PROGRAM REORG2
```

Note that these data items begin in print position 15.

SKELETON B OPTIONS SECTION

The OPTIONS sections is used to specify various processing options, such as the type of program to be generated (in this case, SKELETON B) and the terminal type.

TR

Each option has a keyword, which is followed by one or more values indicating your choice. Each option has a default as shown. The option need be specified only if the default is not appropriate. Options may be entered in any order.

EXAMPLE:

```
100   OPTIONS
200     SKELETON B
300     SEGMENT BATCH1 BATCH2
```

SUMMARY OF OPTIONS FOR SKELETON B

KEYWORD	VALUES	DEFAULT	MEANING
CHAIN	filename	NONE	Name of a program to be run after the TR program is completed (optional)
SEGMENT	filename1 filename2	NONE	Two names to be specified for generated programs or segments
SKELETON	B	Q	Identifies skeleton (required for B)
TERMINAL	ASCII VT100	ASCII	Type of terminal
VALIDATE	ON OFF	ON	Allows suppression of validation rules in file definition. Validation will only occur for PROMPT items

DESCRIPTIONS OF OPTIONS

CHAIN filename

This keyword specifies the name of a program to be executed after the generated transaction programs are completed. The program can be another INTAC program or any other type of program (BASIC, FORTRAN, PASCAL). Use this keyword to set up an automatic processing stream.

EXAMPLE:

```
100   OPTIONS
200         CHAIN UPDATE
```

The program named UPDATE will be run after the transaction program has been completed.

DEFAULT: No chaining

SEGMENT filename1 filename2

The SEGMENT keyword assigns program names to the generated transaction programs. This SEGMENT statement is required when SKELETON B is used because two program files or segments are generated: filename1 is a program to select and sort records; filename2 is a segment containing the LOGIC section code. filename may be any valid RSTS/E or VMS name.

For RSTS/E systems, filename1 may include account number, device specification, and protection code. (The protection code may not be used if the operating system is VMS.) The account number must be included in the first filename following the SEGMENT statement when the translated program will reside in one account and be run from one or more other accounts. The generation of these programs should occur in the account designated on the SEGMENT statement.

EXAMPLES:

```
100   OPTIONS
200         SEGMENT REORG REDIS

100   OPTIONS
200         SEGMENT [240,10]REORG REDIS

100   OPTIONS
200         SEGMENT REORG<60>  REDIS<60>
```

REORG is the name of the program to SELECT and SORT the data. REDIS is the name of the program which will process the LOGIC section. The second example shows the PDP 11/70 account number where both REORG and REDIS will reside (e.g. 240,10). The third example shows the protection code (associated with the PDP 11/70) being set.

DEFAULT: none for sort and break programs because two programs are generated and must be named

SKELETON B

This keyword designates the structure of the program to be generated; in this case, SKELETON B. Since the default is inappropriate, you must always specify SKELETON B for a sort and break file definition.

EXAMPLE:

```
100   OPTIONS
200         SKELETON B
```

DEFAULT : SKELETON Q

TERMINAL ASCII or VT100

Specifies the type of terminal to be used. If you want to take advantage of screen formatting, you must have a VT100 compatible terminal and must specify VT100 in the OPTIONS section. In addition, you must include screen formatting detail in each LOGIC section SHOW and DISPLAY statement.

A file definition that uses screen formatting will run on hardcopy terminals, but terminal display will be preceded by apparently meaningless characters.

DEFAULT: ASCII

VALIDATE ON or OFF

Specifies whether an answer will be validated automatically against file definition edit parameters after it is prompted in the LOGIC section. (Data is always validated for data type.) Use VALIDATE OFF when you want to do your own data validation in the LOGIC section.

Similarly, the LOGIC section has statements for VALIDATE and STORE and the LOGIC statement PROMPT makes available the options VALIDATE, STORE, and NOSTORE. VALIDATE and STORE share the store function. In addition to performing edit parameter checks, VALIDATE also maintains the answer to the prompt (a data item value) until the next prompt for the same item so that you can perform operations on that data item value.

If you want to by-pass the edit-parameter checks, then specify VALIDATE OFF in the OPTIONS section. STORE becomes the default for PROMPT statements. The STORE statement only maintains the prompt answer in the data item until that item is prompted for again. The answer is not subjected to edit parameter checks, but is available for your LOGIC section operations. If VALIDATE ON is specified in the OPTIONS section (or assumed by default), then VALIDATE becomes the default for all prompt statements in the LOGIC section.

EXAMPLE:

```
100    OPTIONS
200          VALIDATE OFF

4000   LOGIC

6030         PROMPT DEPT.NO "Enter old department number"
6040         GOSUB 20200

20200        FIND FILE2 INDEX1 DEPT.NO ERROR= 20300 NOMESSAGE
20300        SHOW "Dept number not in department file"
```

In this example, the PROMPT statement defaults to the STORE option because VALIDATE OFF has been specified in the OPTIONS section. GOSUB sends the program to a subroutine to do validation. The FIND statement causes INTAC to retrieve the DEPT.NO from another file for checking. When no record is found on the file, this use of VALIDATE OFF enables the TR author to send the program to a line with an appropriate message and then to further operations.

DEFAULT: ON

SKELETON B DEFINE SECTION

The DEFINE section (in conjunction with the LOGIC section) allows you to define new data items. For example, define a new item calculated from items in more than one file accessed for the update operation. Or define a data item to be prompted for during an interactive update session. Use DEFINE items in the TR definition LOGIC section as though they were data items in one of the accessed files.

Each line in the DEFINE section defines one data item and is formatted like a data item in an INTAC file (see Chapter 3, CR).

This section is required only when DEFINED items are to be used in a TR program.

FORMAT:

```
itemname type [xx.y] [PROMPT "prompt msg"] [DEFAULT "defltvalue"]
```

where

`itemname`	consists of from 1 to 24 characters. The characters must be alphabetic, numeric, or periods. The first character must be alphabetic. No spaces or other special characters allowed. Names which are INTAC or BASIC reserved words may not be used (See Appendix D for a list of reserved words).
`type`	data type, as used in INTAC files: S character string R real (floating point) number I integer D date field
`xx.y`	print format consisting of two elements xx and y xx total width of printed field including commas, decimal points and sign or parentheses. For a string field this is the length of the string. For a DATE field, this will be either 8 or 10 depending on the print format specified. y the number of decimal places in floating point numbers only. A limit of sixteen digits exists for REAL variables.

`PROMPT ["prompt msg" ]`

This optional prompt message will be displayed as the prompt for this item. Otherwise, the itemname will be used as the prompt. Prompted items must precede unprompted items in the DEFINE section. The maximum length of prompt message is 32 characters. The PROMPT for items in the DEFINE section is performed once only.

`DEFAULT "defltvalue"`

This optional value will be used as the default for this item. If not specified, no default value will be assumed.

TR

DEFINE SECTION EXAMPLE:

```
200   DEFINE
210       NEW.DEPT.NO I 6
```

WHEN TO DEFINE ITEMS

This chapter suggests that you plan the LOGIC section before the DEFINE section so that you will know exactly which new data items you need to DEFINE. As you write the LOGIC section, you can use any valid BASIC variables without including them in the DEFINE section. Variables must be included in the DEFINE section only in two cases: you will prompt for the value of the data item; the variable will be saved from the SELECT section and used in the LOGIC section. (You may include other variables as you prefer.)

DEFINED items should not be used as control variables in the LOGIC section (e.g. do not define a loop counter). For best results, use a BASIC variable (e.g. ICOUNT) when control variables are needed in the LOGIC section.

A maximum of 99 items may be defined in any INTAC TR definition file.

SKELETON B FILES SECTION

The FILES section specifies the INTAC files to be used in the generated program. Any files externally referenced by a file definition must also be specified in the FILES section.

FILE SECTION STATEMENTS

```
FILEn filename [READONLY]
```

This statement specifies a file to be used by the generated program. Up to three INTAC files may be specified, for n=1, 2 and 3. (FILE3 cannot be specified if there are one or more external statements specified. See below, EXTERNAL filename.)

If you will use an INTAC file only to validate or retrieve information (not update). READONLY may be specified. This option protects the file by preventing the user from adding, updating, or deleting records from the file. It is also a useful option when multiple users are allowed.

```
EXTERNAL filename
```

An EXTERNAL filename statement should be used when an external file reference occurs in the INTAC file definition of a file to be accessed. Items from a file defined as EXTERNAL cannot be referenced. If you want to reference items from a file, the file must be specified with a FILE2 statement. (A file need not be listed twice; you do not need to list it with an EXTERNAL statement if you have listed it with a FILEn statement.) There is no limit to the number of EXTERNAL statements in a TR program. The file specified as FILE1 cannot be specified as an EXTERNAL file.

FILES SECTION EXAMPLE:

```
300    FILES
310        FILE1 ASSET
320        FILE2 ORDER READONLY
400        EXTERNAL VENDOR
```

In this example, the INTAC files ASSET and ORDER are being used. Additionally, the INTAC file VENDOR is used as external reference file in the ASSET file and must be specified separately. (The EXTERNAL file VENDOR is used by ASSET as a check to ensure the values entered in ASSET exist in the validation file (e.g. VENDOR). Note that ORDER is being used only to retrieve information and is specified as READONLY.

SKELETON B SELECT SECTION

The SELECT section is used in the SORT/BREAK skeleton to specify the selection and sorting of records and to specify the control break fields. It generates a program which will be run prior to the program generated by the LOGIC section. This program will select the records to be processed, store them in a sort file, call the sort program and cause it to chain to the logic program after sorting.

The SELECT section may contain BASIC statements as well as the commands shown below. (See Appendix B.)

SUMMARY OF SELECT STATEMENTS

The SELECT section may contain the following statements. The GET1 statement is required; the others are optional. The order of these statements is significant as is explained below.

`GET1`
`GET2 INDEXn item1 item2 ... itemn`
`GET3 INDEXn item1 item2 ... itemn`
	Useto specify records

`INCLUDE IF condition`
`EXCLUDE IF condition`
	Use to limit record selection

`SORT item1 item2 ... itemn`	*Use to specify sequence in whichto sort the selected records*
`BREAK item1 item2 ... itemn`	*Useto define break fields*

The index may be any index other than index0. The item name may be any valid INTAC item except item0. Each of these statements is described in detail beginning on page 11-46.

ORDER OF SELECT STATEMENTS

The order of statements in the SELECT section is significant.

The GET1 statement must always be the first INTAC statement in the SELECT section.

INCLUDE or EXCLUDE statements should be placed after the GET statement or statements to which they refer. If no INCLUDE or EXCLUDE statements are specified, every record in FILE1 is selected along with associated records from FILE2 and FILE3 (if GET2 and GET3 are specified).

The SORT and BREAK statements must be the last INTAC statements in the SELECT section. The SORT statement should precede the BREAK statement if both are included in the TR program.

ORDER OF EXECUTION

INTAC reads data from the files in the following manner:

- Each record from FILE1 is read by the GET1 statement.
- FILE1 INCLUDE or EXCLUDE tests are made, and if the record is to be excluded, no reading is done from FILE2 or FILE3.
- If the FILE1 record is to be included, then the corresponding record on FILE2 is read if there is a GET2 statement.
- INCLUDE or EXCLUDE tests following GET2 are executed, and if the record should be excluded, no reading is done from FILE3.
- FILE3 is read and handled in the same manner as FILE2 if there is a GET3 statement.
- If all INCLUDE and/or EXCLUDE tests have been passed, the record is selected and stored in the sort file.

BASIC statements preceding the first GET statement are executed only once before the beginning of the selection process. BASIC statements after the SORT statement are executed only once at the end of the selection process. No BASIC statements may follow the BREAK statement in the SELECT section.

INCLUDE/EXCLUDE EXAMPLES

The following examples illustrate how placement of the INCLUDE and EXCLUDE statements can dramatically affect the run time of your program.

EFFICIENT EXAMPLE:

```
340    SELECT
350        GET1
360        INCLUDE IF EMPL.NO <1000
370        GET2 INDEX1 DEPT.NO
380        GET3 INDEX1 DIV.NO
```

This example illustrates an efficient placement of the INCLUDE statement. Since the INCLUDE test is based on FILE1 only, FILE2 and FILE3 will not be read unless the FILE1 record is to be included.

INEFFICIENT EXAMPLE:

```
340    SELECT
350        GET1
360        GET2 INDEX1 DEPT.NO
370        GET3 INDEX1 DIV.NO
380        INCLUDE IF EMPL.NO <1000
```

In this example, each time FILE1 is read, FILE2 and FILE3 are also read, even if the record is not to be selected. If you were including 20 records out of 2000, you will have performed up to 3960 unnecessary reads on the other 2 files.

SELECT STATEMENTS

The following statements are presented in alphabetical order for reference. The order of SELECT statements in the file definition follows the rules described on the preceding page.

BREAK item1 item2 ... itemn[:n1,n2]

The BREAK statement is used to define the control break fields for the program. The break fields are listed from major to minor break field. The first item will control BREAK1, the second BREAK2, and so on. You can name a maximum of 8 control break fields. The items may be any data items from FILE1, FILE2, or FILE3. BREAK0, the end of program, is automatically defined as the highest level break.

You can specify a part of an item to be a break field. This is done by following the item with :n1,n2 where n1 is the starting position of the break field within the item, and n2 is the length of the break field. Partial fields should only be specified for string data items.

The BREAK statement should be placed following all GET, INCLUDE, EXCLUDE, and SORT statements. A BEGIN and/or END subsection may be included in the LOGIC section for each item listed on the BREAK statement. No BASIC PLUS or VAX BASIC statements may appear after the BREAK statement in the SELECT section. The BREAK statement may be omitted, in which case the entire file will be treated as one group.

BREAK items must have been specified as SORT items; but all SORT items need not be BREAK items. See the following section for further discussion of the usage of control BREAKS.

EXAMPLE:

```
310        BREAK DIVISION DEPT.NO
```

OR

```
310        BREAK DUE.DATE PO.NUMBER:3,2
```

TR

GET1

This statement reads each record from the primary file specified as FILE1 in the FILES section.

GET2 INDEXn item1 item2 ... itemn

This statement specifies an optional corresponding secondary record from FILE2.

INDEXn is used to specify the index number from FILE2 that will be used to access the file.

The items are names of data items from FILE1 that can be used to create a lookup key for INDEXn of FILE2.

EXAMPLE:

Assume FILE1 is an asset file called ASSET.INT which contains an item called DEPT.NO. Also, assume FILE2 is a department file called DEPT.INT which has INDEX1 defined as a department number.

```
340   SELECT
350        GET1
360        GET2 INDEX1 DEPT.NO
```

GET3 INDEXn item1 item2 ... itemn

This statement specifies an optional corresponding record from FILE3. Its format is the same as the GET2 statement. INDEXn should identify an index from FILE3. The items may refer to items in FILE1 or FILE2. If a name is duplicated on more than one file, be sure to use qualified names.

EXAMPLE:

```
340   SELECT
350        GET1
360        GET2 INDEX2 F1.MANAGER
370        GET3 INDEX1 F1.DEPT.NO
```

INCLUDE IF condition
EXCLUDE IF condition

The INCLUDE and EXCLUDE statements are used to limit the records selected from the files. Only one INCLUDE or EXCLUDE statement may occur after each GET statement. Once found and tested (according to the 'condition') the record will be included or excluded as indicated.

The condition consists of 2 values separated by a logical operator. The values may be INTAC data items, data items from the DEFINE section, numbers, character strings in quotes, or any valid BASIC PLUS or VAX BASIC variable.

Partial INTAC data items may be specified using BASIC statements (MID, LEFT, etc.).

The operators are:

<	LESS THAN	
>	GREATER THAN	
=	EQUALS	
<=	LESS THAN OR EQUAL TO	
>=	GREATER THAN OR EQUAL TO	
<>	NOT EQUAL	
==	APPROXIMATELY EQUAL	*(Real variables only)*

Conditions may be joined by AND or OR. Parentheses can be used to form more complicated conditions.

INCLUDE and EXCLUDE statements should be placed immediately following the GET statement or statements to which they refer.

EXAMPLE:

```
     350       INCLUDE IF NAME="SMITH"
     360       EXCLUDE IF NAME="SMITH" AND DEPT >200
```

SORT item1 item2 ... itemn[:n1,n2]

The SORT statement specifies the sequence in which to sort the selected records. Sorts are normally in ascending order; however, you may indicate a descending sort by preceding the item by a hyphen. Enter a list of data items from FILE1, FILE2, or FILE3 that are to be used to sort the records. The items are listed from major to minor sort fields (e.g. item1 is major and itemn is minor).

You may specify a part of a string item to be a sort field. This is done by following the item with :n1,n2 where n1 is the starting position of the sort field and n2 is the length. Partial fields should only be specified for string data items.

If the SORT statement is omitted, the program will use INDEX0 of FILE1, (the logical record number), as the sort sequence.

The SORT statement should be placed after all GET, INCLUDE, and EXCLUDE statements and before any BREAK statement.

EXAMPLES:

```
     340       SORT DEPT.NO PURCHASE.ORD.NO VENDOR.NO
```

or

```
     340       SORT F1.DEPT.NO F2.MANAGER:4,10
```

The first example will sort on (major to minor) DEPT.NO, PURCHASE.ORD.NO, and VENDOR.NO The second example will sort on DEPT (from FILE1), and part of MANAGER from FILE2 (to sort on last name instead of first initial, a field beginning at position four with a total length of ten characters).

TR

SKELETON B LOGIC SECTION

The LOGIC section specifies the processing rules for the selected records. The LOGIC section allows the full flexibility of BASIC statements as well as INTAC statements.

In the definition file, lines 3500 through 19999 are reserved for the LOGIC section.

The Skeleton B LOGIC section is divided into BEGIN, DETAIL, and END subsections describing operations to be performed at control breaks and for each detail record.

SUMMARY OF SKELETON B LOGIC STATEMENTS

BASIC code may be interspersed with LOGIC section statements. LOGIC section statements include the following:

STATEMENT	PURPOSE
ADD	Adds a record to an INTAC file
CLEAR	Clears the line or screen before prompting (VT100 only)
DELETE	Deletes the current record on an INTAC file
FIND	Searches for and retrieves a record with a specific value
FINDNEXT	Retrieves the next record
GOTO	Transfers control of the TR program to a specified line number
POSITION	Moves the VT100 cursor to the specified position on the screen
PROMPT	Defines prompting; has options, including video effects
SHOW	Specifies data items to be displayed and position on screen or page
STORE	Stores the answer just entered by user (used instead of VALIDATE statement or option)
TEST	Tests the answer to a prompt and transfers control of the program based upon the test
UPDATE	Updates the current record
UPDATI	Changes the index for the current data record
VALIDATE	Validates and stores the last answer given

SKELETON B LOGIC SUBSECTIONS

The subsections of the Skeleton B LOGIC section are named as follows:

```
3500      BEGIN PROGRAM
3600      BEGIN BREAKn
3700      DETAIL PROCESSING
3800      END BREAKn
3900      END PROGRAM
```

or

```
3500        BEGIN PROGRAM
3510        BEGIN BREAK itemname
3520        DETAIL PROCESSING
3530        END BREAK itemname
3540        END PROGRAM
```

Only the DETAIL subsection is required. BEGIN and END subsections are necessary only when you wish to indicate special processing.

The format of the BEGIN and END statements is:

```
3500        BEGIN [BREAK] itemname
3600        BEGIN [BREAK] number

4500        END [BREAK] itemname
4600        END [BREAK] number
```

The word BREAK is optional. The itemname must be one of the itemnames on the BREAK statement in the SELECT section. If you use break number, it must be a relative break defined on the BREAK statement. BEGIN and END statements are not both required for a break item.

EXAMPLES:

The following are examples of valid BEGIN and END subsection names.

```
3700        BEGIN BREAK 3
4300        BEGIN 3
4240        BEGIN EMPL.NO
4300        BEGIN BREAK EMPL.NO

3900        END BREAK 3
4500        END 3
3700        END EMPL.NO
3500        END BREAK EMPL.NO
```

There may be up to nine BEGIN and nine END subsections, including BEGIN PROGRAM, END PROGRAM, and one for each control break defined on the BREAK statement in the SELECT section.

All LOGIC statements must follow one of the subsection headers. The following example shows the subsections that could be used for an employee file update.

```
100   SELECT
 .
 .
300         SORT COMPANY DIVISION DEPT EMPL.NO
 .
 .
500         BREAK COMPANY DIVISION EMPL.NO
 .
 .
3600   LOGIC
3650       BEGIN PROGRAM
 .
 .
3700       BEGIN COMPANY
 .
 .
```

```
3800        BEGIN DIVISION
  .
  .
3850        BEGIN EMPL.NO
  .
  .
3875        DETAIL PROCESSING
  .
  .
3900        END EMPL.NO
  .
  .
3925        END DIVISION
  .
  .
3950        END COMPANY
  .
  .
3975        END PROGRAM
```

To refer to the values in the previous record in the END BREAK sections, prefix the INTAC or DEFINED item name with OLD (for example, OLD.EMPL.NO). These OLD items cannot be assigned new values. The old items are not used in file updates.

DETAILED DESCRIPTIONS OF LOGIC STATEMENTS

The following LOGIC statements are described in alphabetical order for easy reference. In addition to these INTAC statements, you may use BASIC statements. For example, you may assign a value to an INTAC or DEFINE item (e.g. DEPT.NO=NEW.DEPT.NO).

ADD FILEn [ERROR = line number] [NOMESSAGE]

Adds a record to INTAC file n as defined in the FILE section. This statement assumes that the processing in some way has filled the record with values for all data items (other than DATE.LAST.EDIT and DELETE.FLAG.) In order to get valid data in each field ALL data items in the INTAC file record must be initialized (other than date last edit and delete flag). If any data items have not been given values, random characters may appear in the updated file.

If an error occurs, control is transferred to the line number specified in the ERROR option. The line number should be a line in the same BEGIN, END, or DETAIL section or in the CODE section. An error message is printed first unless NOMESSAGE is specified. If the ERROR = option is omitted, the program exits on any error. Use the NOMESSAGE option when you want to create your own error-handling procedure. See Appendix B for error-handling information.

EXAMPLE: `ADD FILE1 ERROR = 22000`

```
CLEAR     [ALL ]
          [LINE]
          [SCREEN]
```

For VT100 compatible terminals, you can use this statement to clear the entire screen, clear from the cursor to the end of the current line, or clear to the end of the screen. Use TERMINAL VT100 in the OPTIONS section. The statement will be ignored if the TR program is run on a hardcopy terminal.

EXAMPLES:

```
CLEAR LINE
CLEAR SCREEN
```

DELETE FILEn [ERROR = line number] [NOMESSAGE]

The DELETE statement will delete the current record on INTAC FILEn. If an error occurs, control is transferred to the line number specified. The line number should be a line in the same BEGIN, END, or DETAIL section or in the CODE section. An error message is printed first unless NOMESSAGE is specified. If the ERROR = option is omitted, the program exits on any error. See Appendix B for error-handling information.

EXAMPLE: `DELETE FILE2 ERROR = 23990`

FIND FILEn INDEXn value [ERROR = lineno] [NOMESSAGE]

The FIND statement retrieves a record in the specified file using the specified index. FILEn is the file as specified in the FILE section. INDEXn is the index number in the file definition.

Value can be any INTAC data item name, a DEFINE data item name, a BASIC variable, or an actual value for a data item or variable or, finally, a combination of the above joined with a plus sign (+).

If an INTAC error occurs, control is transferred to the line number specified. The line number should be a line in the same BEGIN, END, or DETAIL section or in the CODE section. An error message is printed first unless NOMESSAGE is specified. If the ERROR = option is omitted, the program exits on any INTAC error.

If you specify INDEX0, the record will be retrieved by record number. When you retrieve by record number, be sure that value is a real number. FIND by record number will transfer control to an ERROR routine (if you use ERROR = lineno) when it finds a deleted record. The value of the error flag (ERROR%) will be -7.

See Appendix B for information on error handling using the FIND command.

EXAMPLES:

```
FIND FILE2 INDEX1 9001 ERROR = 20190
```

This first example will get a record from FILE2 (which in the chapter example is DEPT.INT) with INDEX1 (DEPT.NO) equal to 9001. Control will transfer to line 20190 if there is an error.

```
FIND FILE2 INDEX1 NEW.DEPT.NO ERROR = 20190
```

This second example will get a record from FILE2 with INDEX1 equal to the value in the DEFINE data item NEW.DEPT.NO. This variable could also be a BASIC variable or a data item in one of the accessed files. When an error condition is detected, processing will return to line 20190.

FINDNEXT FILEn INDEXn [value] [**ERROR = lineno.**]

The FINDNEXT statement gets the next record from INTAC FILEn using INDEXn. FINDNEXT must be preceded by a FIND for the same file using the same INDEXn. FINDNEXT cannot be used for INDEX0(record number). The line number should be a line in the same BEGIN, END, or DETAIL section or in the CODE section. If an error occurs, control is transferred to the line number specified. An error message is printed first unless NOMESSAGE is specified. If the ERROR = option is omitted, the program exits on any error.

EXAMPLE: `FINDNEXT FILE1 INDEX3 ERROR = 23990`

POSITION x;y

For VT100 compatible terminals, use the POSITION statement to position the cursor to the specified line y and column x. Use the TERMINAL VT100 in the OPTIONS section. On the VT100 termnal, there are 24 lines and 80 columns.

EXAMPLE: `POSITION 3;7`

This example will position the cursor to line seven (7) and horizontal column three (3) on the line. It is usually easy to estimate the horizontal column number.

PROMPT item [**"custom message for prompt"**] [**option**]

The PROMPT statement is used when you wish to prompt for input data. The shortest form of the statement is: `PROMPT item`

Item may be an INTAC data item name or an itemname from the DEFINE section. Item may be followed by "custom message for prompt " which is your own prompt message to be used in place of the itemname at the time of execution of the program.

PROMPT OPTIONS:

The following options may be added to the PROMPT command. They may be given in any order and should be separated from each other by a space.

CLEAR LINE/SCREEN/ALL
DEFAULT "value"/NODEFAULT
HELP "text"
POSITION x;y
REVERSE/UNDERSCORE
VALIDATE/STORE/NOSTORE

These options are described in detail below in alphabetical order.

CLEAR LINE
CLEAR SCREEN
CLEAR ALL

For a VT100 compatible terminal, you may use this option to clear the entire screen, to clear from the cursor to the end of the line or screen before prompting. Use TERMINAL VT100 in the OPTIONS section.

DEFAULT "value"
NODEFAULT

Specifies a default value to be used in place of any default listed in the file definition. NODEFAULT suppresses use of file default value and forces the user to enter a value. If DEFAULT is specified or the file default is used, the default value will be displayed in the prompt. If you omit this option, the file definition default value (if any) remains in force.

HELP "text"

Specifies a help message to be used when help is requested in response to prompt message by entering ? or HELP.

POSITION x;y

For VT100 compatible terminals, this option positions the prompt message at the VT100 screen position column x, line y (e.g. POSITION 1;5 means put the cursor in position 1 and line 5 of the VT100 terminal screen). You may abbreviate the keyword POSITION to POS. Use TERMINAL VT100 in the OPTIONS section.

REVERSE
UNDERSCORE

For VT100 compatible terminals, this option causes the space for the prompt answer to be displayed in reverse video or as an underscore. This special screen formatting instruction is available only when you have given the TERMINAL VT100 option.

VALIDATE
STORE
NOSTORE

This option indicates whether or not you want INTAC to validate and store the answer automatically after prompting. Validation is done using the edit parameters in the file definition for this item. VALIDATE validates and stores the answer temporarily in the item. STORE stores the answer without validating. NOSTORE indicates that the answer will be neither validated nor stored. Use the STORE and NOSTORE options when you want to specify your own validation routines. If you use NOSTORE, you will normally be doing your own testing and validating.

If you omit these options, the default depends upon your specification in the OPTIONS section of the TR definition file.

PROMPT EXAMPLES:

```
PROMPT VENDOR.NO "Vendor Number" VALIDATE
```

In this example the INTAC item VENDOR.NO will be prompted for with the message "Vendor Number?". If there is a default value in the file the message will be "Vendor Number <default>?". The value entered will be validated using the MINIMUM, MAXIMUM, TABLE, or EXTERNAL FILE specified in the file definition for this item. If the entered value is not valid, the prompt will be repeated. If it is valid, the entered value will be stored in the item.

```
PROMPT VENDOR.NO POS 1;5 CLEAR SCREEN UNDERSCORE NOSTORE
```

In this example the INTAC item VENDOR.NO is prompted for with the message "VENDOR.NO" A file default would appear as "VENDOR.NO <default>?". The screen is cleared before the prompt message is printed. The prompt message will begin at line 5, column 1 on the screen. The space for the answer will be underlined. The entered value will not be validated or stored. The PROMPT options POS, CLEAR SCREEN, and UNDERSCORE in this example will be ignored if the terminal in the OPTION section is ASCII.

SHOW element1 element2 ... elementn [+]

The SHOW statement is used to specify the data elements to display. The elements may be INTAC data items, DEFINED items, BASIC variables or a text message enclosed in quotes. The SHOW statement may also specify a print position across the screen or page and a format. When using video effects for VT100 terminals, you must specify print position in every SHOW statement.

If a SHOW statement ends with a plus sign (+), the print line is to be continued on another SHOW statement.

For file definitions in which no special video effects are used, each element in the SHOW statement may take the form:

```
itemname[:positionn][,format]
"text message"[:positionn][,format]
```

For file definitions using video effects and specifying TERMINAL VT100 in the OPTIONS section, each element in the SHOW statement can take the form:

```
itemname[:positionn;z][,format]
"text message"[:positionn;z][,format]
```

where

`itemname`	any INTAC data item name, DEFINE data item name, or BASIC variable or constant
`"text message"`	Any text message
`(nonvideo)` `positionn`	For hardcopy terminals, a specific print position on the line. positionn may be a number (from 1 to the maximum width of your terminal) or an expression that can be evaluated at execution time. If a position is omitted, the item will print in the next available print position. If the VT100 format is being used, refer to the special position statement (below) for cursor movement.
`(VT100 format)` `positionn;z`	positionn;z is a specific print position on the screen. positionn is a number from 1 to the maximum width of your terminal-- usually 80 or 132) indicating the character position across the screen. z is a number from 1 to 24 indicating the line on the screen. The print position is required if the VT100 option is specified in the OPTIONS section.
`format`	xx.y is an INTAC print format. If it is omitted, the format is taken from the item definition. If the position is omitted and the format is included, the format must be preceded by a comma. xx is the width of printed field including commas, decimal point and sign. y is the number of decimal places.

SHOW EXAMPLES:

```
SHOW VENDOR.NO VENDOR.NAME TOT.AMT
```

In this first example, each of the data items will be printed in the next available print position. All of the elements except for TOT.AMT are data items in the INTAC file and INTAC will print them from the file. TOT.AMT was defined as a new data item and used in the LOGIC section to hold the results of a calculation. The calculated results will print.

```
SHOW PURCHASE.ORD.NO:1 DESCRIPTION:10 ASSET.TYPE:33,2.0  +
SHOW VENDOR.NO:40 LIFE:48,10.2
```

In the above second example, the first data element is to be printed in position 1. The second element (DESCRIPTION) will be printed in position 10 on the line using its

INTAC file default print format. The third element, ASSET.TYPE, will be printed in position 33 on the line, using a format of 2.0. The two fields defined in the second SHOW statement will print on the same line in print positions 40 and 48, respectively.

```
SHOW "Vendor #" F1.VENDOR.NO:12 "  not on VENDOR.INT"
```

This third example will print the text beginning at print position one and the INTAC item F1.VENDOR.NO beginning at position twelve followed in the next available print position by the text.

STORE

This statement indicates that you do not want INTAC to validate the answer automatically after prompting. Validation is usually done using the edit parameters in the file definition for this item. STORE holds the answer without validating it. If you use STORE, you will normally include statements to do your own validating. When you omit VALIDATE or STORE statements, the default depends upon your specification in the OPTIONS section of the TR definition file.

EXAMPLE: `PROMPT F1.VENDOR.NO "Vendor number" STORE`

TEST operator "expression" THEN lineno

The TEST statement tests the answer to the last PROMPT statement ("expression") and goes to the specified line number if the test is true. Operator must be a comparison operator (>, <, >=, <=, =, <>, ==) and the answer must be a string-valued expression. The answer in string form is compared with this expression. The line number should be a line in the same BEGIN, END, or DETAIL section or in the CODE section.

EXAMPLE:

```
PROMPT ACTION "ENTER UPDATE,DELETE,STOP"
TEST = "STOP" THEN 32000
TEST = "UPDATE" THEN 3800
TEST = "DELETE" THEN 4000
```

In this example, the BASIC variable ACTION receives a value through prompting. That value is then compared to "STOP", "UPDATE", and "DELETE".

UPDATE FILEn [ERROR=lineno] [NOMESSAGE]

The UPDATE statement will change the current record on INTAC FILEn. If an error occurs, control is transferred to the line number specified. The line number should be a line in the same BEGIN, END, or DETAIL section or in the CODE section. An error message is printed first unless NOMESSAGE is specified. If the ERROR = option is omitted, the program exits on any error. UPDATI should be used after UPDATE if an index value has changed.

EXAMPLE: `UPDATE FILE2 ERROR = 32000`

UPDATI FILEn INDEXn old.value [ERROR =lineno] [NOMESSAGE]

The UPDATI statement changes the specified index for the current data record from the old value given to the new value formed from the current value of the items in the record. The line number should be a line in the same BEGIN, END, or DETAIL section or in the CODE section. An error message is printed first unless NOMESSAGE is specified.

This statement is used after FIND and UPDATE and BASIC assign statements if an index item changes. Note that if an item is used in multiple indexes, then UPDATI statements must be given for each index.

EXAMPLE: `UPDATI FILE1 INDEX3 VALUE ERROR = 23990`

VALIDATE

The VALIDATE statement validates and stores the answer given to the last prompt. Validation is done using the edit parameters in the file definition for this item. This statement is normally used when the NOSTORE option is specified in a PROMPT statement. It is used in conjunction with specification of VALIDATE OFF in the OPTIONS section.

EXAMPLE: `VALIDATE`

TR

SKELETON B CODE SECTION

The CODE section is for user subroutines and functions. Lines 20000 through 23999 are reserved for the CODE section. The line numbers specified in the CODE section (unlike the line numbers in other sections) are not changed by the transaction program generator.

The CODE section may contain any BASIC or INTAC TR LOGIC statements and variables, except the PROMPT statement.

The CODE section may only be referenced from the LOGIC section. Control is transferred to the CODE section from the LOGIC SECTION in one of the following ways:

- function call in a BASIC statement
- GOTO or GOSUB in a BASIC statement
- ERROR = line number in a LOGIC statement.

If the CODE section is used, it must be the last section in the report definition file. See Appendix B for error-handling information.

TRANSACTION PROGRAM GENERATION

Once you have created a definition file, you must translate the definition into a BASIC program. Then you may run the program whenever you need to update the file.

TR COMMAND

The INTAC TR command is used to generate a BASIC PLUS or VAX BASIC transaction program from a definition file. Here is an example of the dialogue when the TR command is given.

```
COMMAND? TR

GENERATE AN INTAC TRANSACTION PROGRAM

DEFINITION FILE? TRQSIM.DEF

GENERATE A QUESTION LOOP

258 LINES, FILE: USR:TRQSIM.BAS
BEGINNING COMPILATION OF SEGMENT: USR:TRQSIM(BP)

COMPILE CONTINUING
END OF COMPILE GENERATED TRANSACTION PROGRAM

COMMAND?
```

The TR command asks the following question:

```
DEFINITION FILE? filename/options
```

Enter the name of your transaction definition file, followed by the options selected. An extension of .DEF is assumed if none is given. INTAC will create a file called progname.BAC (or progname.TSK) which is the generated transaction program on a RSTE/S system or progname.EXE which is generated on the VAX (see SEGMENT statement in OPTIONS section). On a RSTE/S system, two separate programs will be created if SKELETON B is being used. In response to your command, RUN filename1, INTAC runs both programs in the order they are specified in the SEGMENT statement (e.g. SEGMENT BATCH1 BATCH2 would generate two programs which will be run in the order BATCH1, BATCH2). For more information, see detailed explanation of OPTIONS keywords, SKELETON B.

If a generated program with the same name already exists, you will be asked if you want to delete it. If your response is NO, the TR will not be generated.

Occasionally, the TR process will display a warning message (preceded by a percent sign) generated by the BASIC compiller. For example, a common message is `"% inconsistent function usage at lineno."` These warning messages do not interfere with the TR process and may be ignored. If a serious problem exists in the TR, an error message will be displayed and the process will not proceed.

OPTIONS

Any logical combination of options may be specified. Options are separated from the filename and from each other by a slash. If no options are specified the default is to compile the program in BASIC PLUS or VAX BASIC and not run the program.

The available options are:

RUN	run the generated program(s) immediately after they are generated and compiled
NOK	do not kill source code (progname.BAS) (normally the .BAS file is deleted when the generation is completed.)
BP2	on the PDP 11's, compile in BASIC-PLUS-2 (creates progname.TSK rather than progname.BAC) (default is BASIC-PLUS)
NCO	do not compile the generated source code
NOS	do not append skeleton code (for debugging purposes)
KB:	output generated code to terminal (should use /NOS too for debugging purposes)
DIS	display the commands used to compile the program

EXAMPLE: `DEFINITION FILE? **SALES/BP2/NOK**`

The above example will translate the file SALES.DEF into a BASIC program and then compile in BASIC-PLUS-2 creating SALES.TSK. It will also preserve the BASIC code in a file called SALES.BAS.

SUBSEQUENT PROCESSING

Once the BASIC program has been generated from the transaction definition file you may run the program by entering the command: `RUN programname`

Programname is the name specified on the SEGMENT statement in the OPTIONS section. For a sort and break program, this is the name of the sort segment. If no segment name was specified for a question loop program, programname is the same as the name of the definition file.

EXAMPLE:

```
RUN TRQSIM

TRANSACTION PROGRAM TRQSIM

Enter purchase order number ? 10053
Vendor number? 10
Order entered
Enter purchase order number ?
```

REFERENCE WORKSHEET FOR Q SKELETON TR

Use the following worksheet as a reference as you write your own Q TR programs.

```
11     !*****************************************************************
12     !TR SKELETON Q FILE NAME:
13     !PURPOSE of TR:
14     !AUTHOR:
15     !DATE:
16     !****************************************************************
17     !
100     OPTIONS

110         SKELETON Q
120         SEGMENT segmentname                          (optional)
130         TERMINAL ASCII or VT100                      (default ASCII)
140         VALIDATE ON or OFF                           (default ON)
150         CHAIN filename                               (optional)


200     DEFINE

        itemname type [xx.y][DEFAULT "defltvalue"]



300     FILES
310            FILE1 filename


               FILE2 filename                         (optional)


               FILE3 filename                         (optional)



               EXTERNAL
```

```
3500   LOGIC

5000       BEGIN PROGRAM
6000       Q10
```

In the LOGIC section, use any of the following statements in a question structure.

```
ADD FILEn [ERROR = Qn ] [NOMESSAGE]
          [ERROR = lineno ]
CLEAR [ALL]
      [LINE]
      [SCREEN]

DELETE FILEn [ERROR = Qn]  [NOMESSAGE]
             [ERROR = lineno]

EXIT
FIND FILEn INDEXn value [ERROR = Qn]  [NOMESSAGE]
                        [ERROR = lineno]

FINDNEXT FILEn INDEXn [value]  [ERROR = Qn]
                      [ERROR = lineno]

GOTO [Qn]
     [lineno]

POSITION x;y
PROMPT item ["custom message for prompt"]  [option]
  whereoption maybe any of the following:

    BACKSLASH Qn
    CLEAR LINE/SCREEN/ALL
    DEFAULT "value"/NODEFAULT
    EXIT "value"/NOEXIT
    HELP "text"
    POSITION x;y
    REVERSE/UNDERSCORE
    VALIDATE/STORE/NOSTORE

SHOW element1 element2 ... elementn [+]
   whereeach element can take the form

    itemname:position,format
    'text message':position,format

STORE
TEST operator "expression" THEN [Qn]
                                [lineno]

UPDATE FILEn [ERROR=Qn]  [NOMESSAGE]
             [ERROR=lineno]

UPDATI FILEn INDEXn old.value [ERROR =Qn]  [NOMESSAGE]
                              [ERROR=lineno]

VALIDATE

19000      END PROGRAM

20000   CODE
20100 !*****************************************************************
20110 !***      USER SUBROUTINES
20120 !*****************************************************************
```

REFERENCE WORKSHEET FOR B SKELETON TR

Use the following worksheet as a reference as you write your own TR programs.

```
11 !***************************************************************
12 !TR SKELETON B FILE NAME:
13 !PURPOSE OF TR:
14 !AUTHOR:
15 !DATE:
16 !***************************************************************
17 !
100    OPTIONS
110          SKELETON B
120          SEGMENT segmentname1 segmentname2
130          TERMINAL ASCII or VT100 (default ASCII)
140          VALIDATE ON or OFF (default ON)
150          CHAIN filename (optional)

200    DEFINE
```
Include new data item names in the format

```
   itemname type  [xx.y]  [PROMPT "prompt msg"]  [DEFAULT "default value"]

300    FILES
310         FILE1 filename

            FILE2                                      (optional)

            FILE3                                      (optional)

            EXTERNAL


340    SELECT
350          GET1

             INCLUDE

             EXCLUDE

             GET2 INDEXn itemm

             GET3 INDEXn itemn

             SORT item(s)

             BREAK item(s)
```

TR

```
3500   LOGIC
```

Use any of the following statements in a SORT and BREAK structure as indicated below:

```
   ADD FILEn [ERROR = lineno]  [NOMESSAGE]
   CLEAR [ALL]
         [LINE]
         [SCREEN]

   DELETE FILEn [ERROR = lineno]  [NOMESSAGE]

   FIND FILEn INDEXn value [ERROR = lineno]  [NOMESSAGE]
   FINDNEXT FILEn INDEXn [value] [ERROR = lineno]  [NOMESSAGE]
   POSITION x;y
   PROMPT item ["custom message for prompt"]  [option]
```

where option may be any of the following:

```
        CLEAR LINE/SCREEN/ALL
        DEFAULT "value"/NODEFAULT
        HELP "text"
        POSITION x;y
        REVERSE/UNDERSCORE
        VALIDATE/STORE/NOSTORE

   SHOW element1 element2 ... elementn [+]
```

where each element can take the form

```
        itemname:position,format
        "text message":position,format

   STORE
   TEST operator "expression" THEN  [lineno]
   UPDATE FILEn [ERROR=lineno]  [NOMESSAGE]
   UPDATI FILEn INDEXn old.value [ERROR =lineno] [NOMESSAGE]
   VALIDATE

5000     BEGIN PROGRAM

6000     BEGIN ITEM1

7000     BEGIN ITEM2

12000    DETAIL PROCESSING

17000    END ITEM2

18000    END ITEM1

19000    END PROGRAM

20000   CODE
20100 !**********************************************************************
20110 !*** USER SUBROUTINES
20120 !**********************************************************************
```