TRIMreport Reportwriter

May 30, 2017

Trifox Inc.

www.trifox.com

Trademarks

Copyright

Contents

Preface 1

Background 1

Support 2

1 Introduction

Defaults 4

Screen Layout 4

Actions and Function Keys 4

Screens 5

2 Designer Elements

Page Formatting 6

Report Blocks 7

Block organization 7

Pagetext 8

Turning the Page 8

Block Detail 9

SELECT Control 9

WHERE Control 10

Trigger Control 11

Showing Other Information 11

Variables 11

Functions 11

Triggers 12

MAIN Trigger 13

User Triggers 13

Break Sub-Block 14

Break Sub-Block Header & Footer 15

Break Control Triggers 15

Special Break Techniques 15

3 Design Considerations

Effective Designs 17

Report Design Methods 17

Distributing Work 17

SELECT-driven Method 18

Default Behavior 18

Examples 20

List-Driven Method 23

Examples 24

Trigger-Driven Method 27

Examples 28

4 Designing a Report

Step by Step 32

Printing Your Report 43

Controlling the Printer 43
Saving Escape Codes 43
Advanced Procedures 44
Calling Other Reports or Applications 44
Making a Report Database-Independent 44
Specifying a Single Pagetext Header/Footer 45
Fields to Center Headings 46
Transforming Values 46
Multiple Columns from One Field 46
Second Tutorial: Two-Block Report 47

5 Running Your Report

Running/Printing Reports 50
Output Buffer 51
Output Files 52
Parameters 52

6 TRIMreport Screens

BLOCK COMMENTS 55
CHOOSE REPORT 56
CHOOSE REPORTBLOCK 58
CHOOSE TRIGGER 60
Predefined Triggers 60
DEFINE BREAK 62
DEFINE FIELD 63
DEFINE PAGETEXT 64
DEFINE REPORTBLOCK 65
DEFINE SELECT 66
DEFINE TRIGGER 68
DEFINE WHERE 69
Screen Painter 70
Main Screen 70
Edit Screen 71
SQL COMMAND 73
SYSTEM COMMAND 74

Appendix A Format Masks **75**

char 75
numeric 75
datetime 76
User-defined 77

Index 79

Preface

Background

Trifox Inc. has been serving the relational database market since 1984 through consulting and the development of software products. In 1987, Trifox created SQL*QMX for Oracle. This easy-to-use, powerful querying and report writing tool, which is based on IBM's QMF, continues to be used at thousands of sites. In 1989, Trifox created TRIMtools, a family of application and reportwriting tools now known as DesignVision. DesignVision was developed in response to the OLTP requirements of several large application vendors.

Database Access

VORTEX is an integrated family of products that allows nearly any production application to access SQL data:

- On any or all of the major relational databases.
- Across networks.
- Across platforms.
- With a dramatic increase in the number of concurrent users.
- Without any additional hardware.

In a client/server or multi-tier configuration, VORTEX makes it possible for your SQL applications to access data on different platforms over one or more network configurations. Currently it supports only TCP/IP.

Inherent in this approach are services that allow production applications originally written for one relational database (such as Oracle) can access the same data on another database (such as Informix), even if it is spread across different databases.

VORTEX Precompilers for C and COBOL, as well as a variety of program interfaces, allow existing SQL programs to take full advantage of VORTEX services such as performance enhancement, transaction monitoring, and flat-file database access.

With VORTEXaccelerator in your configuration, you dramatically increase the number of concurrent users who can log on to a specific SQL production application. Your users experience faster performance and you won't have to change any programs or add any hardware.

Application and Report Development

DesignVision DVapp lets you design, generate, and maintain forms-based applications. You can easily port the pop-up windows, customizable menus and submenus, and custom keyboard assignments, in fact the entire application, to Windows .NET, Unix, OpenVMS, or HTML5 with no extra effort.

The reportwriter, TRIMreport, lets you create simple reports quickly, or complex reports with absolute confidence in their power.

When you want to write stand-alone applications (including triggers) without a user interface, the TRIMpl 4GL language gives you the freedom you want. The procedural language has over 100 database-specific functions that help you write powerful applications in very little time.

Reaching Legacy Data

GENESISsql is a SQL processor that accesses low-level data sources such as ISAM, SDMS, ADABAS, RMS, and MicroFocus and makes the data accessible to VORTEX clients. You can add GENESIS data sources to a VORTEX system in a matter of days, simplifying what used to be an enormous task.

Conventions

Screen shots in this manual come from the Windows version of our software.

Trifox documentation uses the following conventions for communicating information:

Example	Describes
`CHOOSE REPORT > [F3] >`	Press [F3] on the CHOOSE REPORT menu and ...
Right-click	Clicking the right mouse button.
`connect_string`	Replace italicized text with your own variable.
Left-click	Clicking the left mouse button.
`vtxnetd`	Text in bold typewriter style represents strings that you type exactly as they appear in the manual.

Support

If you have a question about a TRIFOX product that is not answered in the documentation (paper or online), contact the Customer Support Services group at:

- support@trifox.com
- Trifox Customer Support Services
  2959 Winchester Boulevard
  Campbell, CA 95008
  U.S.A.
- 408-796-1590

Chapter 1 Introduction

TRIMreport is a 4GL that allows you to create basic to complex reports from data stored in your database.

TRIMreport is completely interactive and includes a screen painter that lets you design the report on screen. You can arrange and edit all the report elements — columns, headings, headers/footers, summary information, and page breaks. You can execute the report design from TRIMreport at any time to validate the design as it develops.

TThe report design is filed and other users can run the report with the runtime version of TRIMreport.

Because TRIMreport shares a library of functions with its companion DVapplication, the two can work seamlessly to create applications and reports that meet your needs. The library makes it possible for you, a designer, to create applications and generate reports without writing any code. At the same time, professional developers can create additional functions, store them in the library, and you can use them to maintain style and functionality consistent with your organization's unique business needs.

TRIMreport lets you write reports as simple or as complex as you like. In its most basic use, you can create a report with the provided defaults. The more advanced levels allow you to build your own definitions of pages, assign actions in a report, and store them in the default library, or *data dictionary*, for future use.

TRIMreport consists of three main parts, all of which operate independent from the database:

- ***Designer*** — creates ascii files that are easily ported across platforms.
- ***Generator*** — creates binary files for specific operating environments.
- ***Runtime*** — executes the report.

This independence lets you control the processes that generate reports, such as how often a report is required to interact with a database — especially important in environments with many users.

The reports typically retrieve data from the database once, and store it locally in lists. To minimize database interaction, TRIMreport uses list management to process additions, validations, modifications, and updates locally.

A TRIMreport report consists of *report blocks*, which contain SELECT statements, text/header/footer areas, fields, and triggers necessary to generate the report.

The report blocks are organized in a left-right/top-bottom structure. Left-right neighboring report blocks are siblings; top-bottom neighboring report blocks are parent-child. Sibling report blocks all have the same single parent. While the report is being designed, only the report blocks with direct parent-child relationships are displayed.

The screen painter never displays siblings on the same screen.

Defaults

TRIMreport comes with a default library that helps maintain consistency in your organization. The functions and defaults are completely under your control — you can use them as-is, or you can replace or modify them, or you can create new ones.

TRIMreport draws from an extensive set of default settings and functions that allow you to write reports without writing any code. Functions for each step of the formatting are added in the design module; the defaults appear on the screen as you design the report by selecting from prompted actions. Once the initial design is created from the defaults, you can use the screen painter to cut and paste elements to meet specific design requirements.

In addition, TRIMreport lets you create defaults that can be used from report to report for uniformity of design. For example, you may want the same header style for the first page of all reports, or you may want certain confidential information, when included in a report, to automatically add a page label indicating the security level.

The salary field can be coded for certain employees as *confidential* depending on the user identification used — the word *confidential* is printed in the report when the employee's data condition is met. When the salary field is created in the database, code for the field is stored in the TRIMreport data dictionary.

Screen Layout

TRIMreport consists of several character-based windows (screens) that let you perform certain functions. Most windows have *actions* that perform operations specific to the dialog; the *function keys*, listed at the bottom of the screen, perform operations that are general in nature.

Windows that appear on the screen, such as the CHOOSE REPORT dialog above, handle your input and initiate an action. The keys for available functions appear at the bottom of the screen. The status line displays messages associated with any attempted procedure; for example, the screen above displays the `OK CONNECT` performed message that appears after you have successfully connected to the database.

The bottom line of the status line displays the current block name and type that is being modified, such as BLOCK (Detail). In the screen painter, the status line includes the current row and column of the cursor.

Actions and Function Keys

To execute an action, put the cursor in the action name and press [Enter]. You can move the cursor using the arrow or [Tab].

Pressing [Tab] once moves the cursor to the next valid field; pressing SHIFT+[Tab] moves the cursor to the previous valid field.

Function keys are listed immediately below the screen work area. All screens have at least one active function key, [F3]. To execute the function key command, simply press the function key that corresponds to the command.

In addition to the eight function keys listed on the screen, two other functions are available: [F9] and [F10].

These keys, as well as [PgUp] and [PgDn], let you move across screens that permit scrolling.

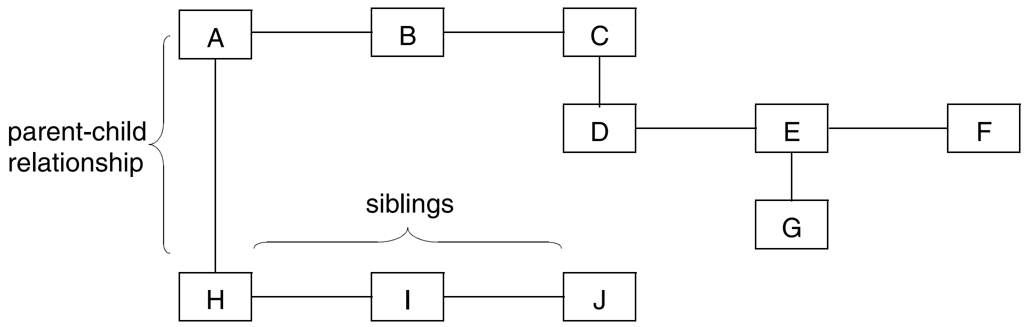
NOTE: *The actual keyboard mappings for function keys are defined in the termtype.key file for your terminal by the GETKEY utility. This section documents the defaults as shipped.*

From any text editing window, such as a trigger window, or from the screen painter, press HOME to exit to the first TRIMreport window, CHOOSE REPORT.

Screens

A TRIMreport report consists of *report blocks*. Report blocks contain SELECT statements, text/header/footer areas, fields, and triggers necessary to generate the report.

The report blocks are organized in a left-right/top-bottom structure. Left-right neighboring report blocks are siblings; top-bottom neighboring report blocks are parent-child. Sibling report blocks all have the same single parent. While the report is being designed, only the report blocks with direct parent-child relationships are displayed. The screen painter never displays siblings on the same screen.



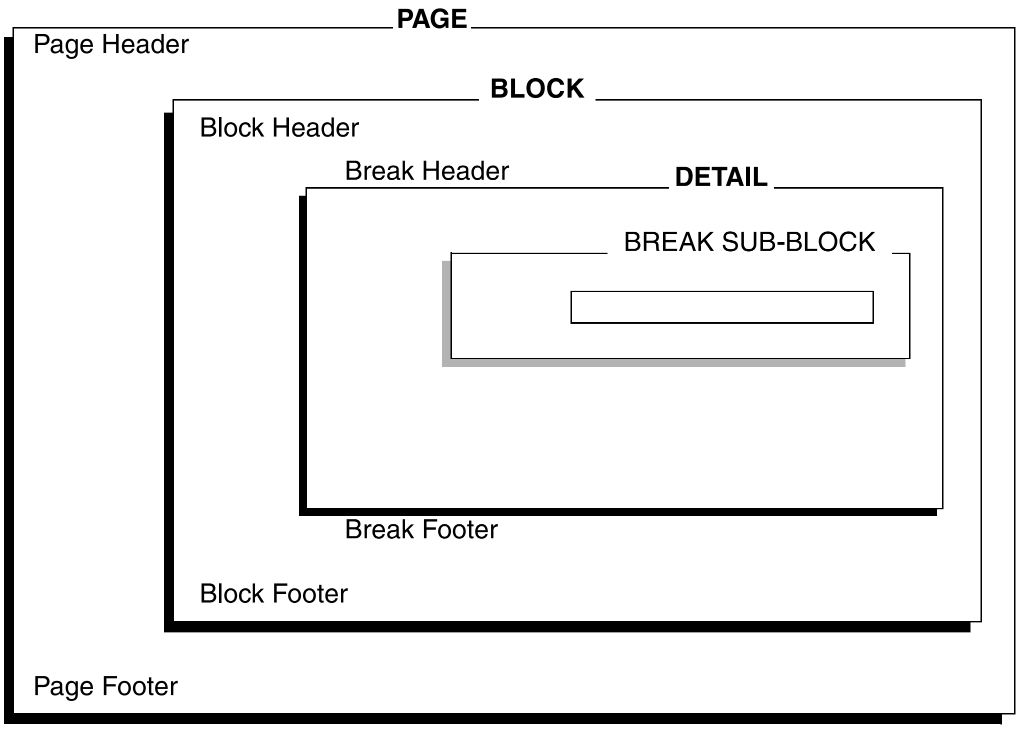
For example (see the diagram above), when report block H is being designed, the screen painter displays A and H. When report block F is being designed, the screen painter displays C and F. Finally, when report block G is being designed, the screen painter displays C, E, and G.

At report runtime, the report block execution sequence is from top to bottom, and left to right. In the above example, the order is A-H-I-J-B-C-D-E-G-F. You can modify the sequence by editing the report block triggers, such as the CHILD and POST-BLOCK triggers in the TRIMreport library, and using the `block()` function.

Chapter 2
Designer Elements

You can have TRIMreport create default designs for you to work with. To create your custom design, you modify elements in the design, and where necessary, add new ones. This chapter describes the elements you'll be modifying and gives guidance on how changes affect your report. You can enhance and edit the report design on screen.

A simple graphic summary of report elements looks like this:



Headers and footers for pages, blocks, and breaks are all optional. Break subblocks themselves are optional and blocks may have multiple break subblocks, which would change this illustration. In addition, blocks can have siblings and children, and children can be parents of other children. Each sibling and child can have its own headers and footers as well as detail and break definitions.

Page Formatting

Blocks (and their components) fit inside pages. The output buffer, which becomes the printed report, is defined by the PAGE LENGTH (number of lines) and PAGE WIDTH (number of characters on a line) fields on the CHOOSE REPORT window. Although the maximum values allowed are 999 lines and 999 characters, it only makes sense to format pages that fit on your paper.

When the paginate trigger is invoked and/or your report calls a `paginate(break)`, the command writes the output buffer to a file and issues a form feed. It then clears and resets the output buffer.

NOTE: *You can suppress or change form feed by passing an option,* -f*, to the report when running it. For details, see "**Running/Printing Reports**" on page 50.*

Report Blocks

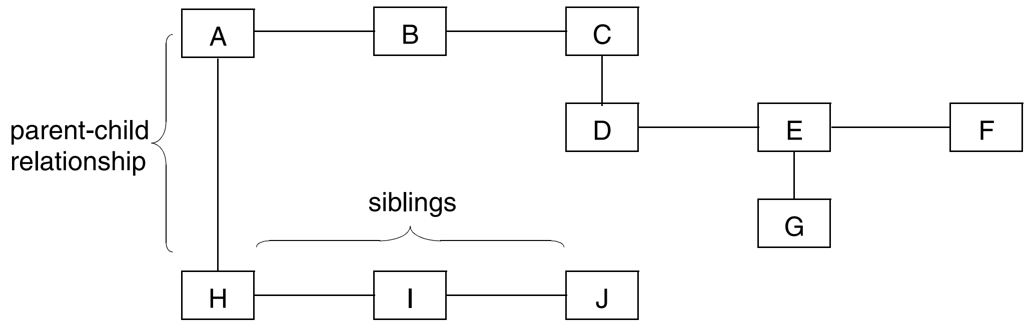
A *report block* is the basic unit for printing or displaying data in the report. Called from the report design, a block has three visual "parts" and up to three control sections. Control sections, made of SQL and TRIMpl code, determine the block's actions and activities.

The visual parts, *header*, *footer*, and *detail area*, are what you see when the report prints out. Block headers and footers, which print once each time a report block is called, contain *text*, printed exactly as you enter it, and *fields*, which are filled in at runtime.

Block organization

Your report processes blocks from left to right and top to bottom. Report blocks that are next to each other are called *siblings*. Siblings all have the same single parent and do not appear together on the screen painter. Report blocks that are above each other are considered to have a *parent-child* relationship. You can only request a child block if its parent has a SELECT statement. Also, you cannot create a child block that causes the combined size of the parent and child to exceed the defined page length.

If you were viewing blocks from the following diagram, when you are working on report block H, you could see A and H. When you work on report block F, you can view C and F. When you work on report block G, you can see C, E, and G.



When you execute a report, the execution sequence is from top to bottom and left to right. In the illustration, the order is A-H-I-J-B-C-D-E-G-F. To modify the sequence, edit report block triggers like CHILD and POST-BLOCK to include the TRIMpl `block()` function. (For details see "***Triggers***" on page 12.)

When you create a block as a child, it inherits pagetext from its parents. As a visual aid, the designer displays a child block positioned within its parent block.

Pagetext

Even though the *page* is a larger element, its appearance is determined by settings in the *block*. Each block has associated specifications for page-level headers and footers that it inherited from its parent block, or that you have modified (CHOOSE REPORTBLOCK > Choose a block > PAGETEXT). This *pagetext* includes three items for building a page around the report blocks:

- Pagetext header
- Pagetext footer
- PAGINATE trigger

Turning the Page

The PAGINATE trigger controls pagetext activity, including creating new pages. It contains code, usually a `paginate()` call, which can have up to three parameters:

- ***header*** — tells the function to insert pagetext header text and fields at the top of a report page.
- ***footer*** — places the pagetext footer text and fields at the end of the report page.
- ***break*** — tells the function to end the current page.

If you create a report block with a two-line pagetext header and a PAGINATE trigger that contains

```
paginate (header);
```

when you run the report, the two-line pagetext header appears at the very top of the page.

Once the header is placed, the variable `G.LINENUMBER` increments by the number of lines in the pagetext header.

As detail lines are written to the output buffer, the runtime executable (TRIMrun) keeps track of `G.LINENUMBER` progress. When the variable reaches the most number of lines that can fit on a page given the specified page length and the number of lines specified in the footer

```
(G.LINENUMBER = PAGE LENGTH - NUMBER OF LINES IN PAGTEXT (FOOTER))
```

TRIMrun calls the PAGINATE trigger again. Typically, the trigger writes the pagetext footer, creates a page break (forces the current buffer to write to the report file or device opened by the last `open()` call, and writes a pagetext header to the next page).

The default PAGINATE trigger, unless modified, performs those three actions in that order when it's called. If you change the trigger, be sure that you force a break before the end of a page. Otherwise, your report returns an error.

NOTE: *G.LINENUMBER and G. PAGEOFFSET only affect detail area text. You cannot use them to move a field in the pagetext text areas.*

Block Detail

The block detail is where the real content of your report is defined. Control sections in the block detail — SELECT, WHERE, and trigger — determine how the detail is completed:

- A SELECT control retrieves data from a database.
- A WHERE control (not to be confused with the WHERE clause in a SQL statement) filters that data, evaluating to either false (0) or true (non-zero).
- Trigger controls (or simply triggers) control the flow of the block's actions.

Block *detail areas* are completed when a SELECT control retrieves data from the database and the WHERE filter passes it on. Generally, the block detail area prints once for each row returned by the SELECT statement.

SELECT Control

The SELECT control can contain either:

- A SQL SELECT statement for example:

  ```
  SELECT * FROM table
  ```

- A list reference. For example:

  ```
  [G.DATA_LIST, ovtvar1, ovtvar2]
  ```

  When the SELECT references a list, every row from the current position to the end of the list is copied to the output buffer. Any WHERE control you've put in the block detail is simply ignored.

Using SELECT

The SELECT control contains the SELECT statement that retrieves data from the database. Each retrieved row that satisfies the WHERE control places a detail area in the report. The statement must be a valid SELECT — an UPDATE, INSERT, DELETE, or other non-SELECT statement causes an error.

Generally, the SELECT has a WHERE clause. For example:

```
SELECT name, dept
   FROM staff a
   WHERE a.dept = &G.DEPARTMENT
```

The example returns rows from two columns (*name* and *dept*) when the *dept* value matches the value of `G.DEPARTMENT` in the main trigger. Notice that the SQL statements are not terminated with semicolons (;).

When you specify a SELECT statement, you must also choose one of the actions on that window — CREATE, DEF_HORZ, DEF_VERT, or DEF_VER2 — for the SELECT to be accepted. When one these actions executes, it creates field structures that correspond to the columns referenced by the SELECT statemenT. The CREATE action builds *field structures*. DEF_HORZ, DEF_VERT, or DEF_VER2 actions build default fields in addition to the field structures.

If the example builds two field structures, *names* and *dept*, you can reference the value of *name* or *dept* for the current row returned from the database as *name* and *dept* (or *block.name* and *block.dept*, where block is the name of the block).

If the database table changes or the columns themselves change, to avoid errors you must issue a CREATE to recreate the report's field structures. The following error:

```
Error: GEN vs RUN internal mismatch
```

means that you must recreate the field structures before running the report against the changed table. You can run a report against a different database than the one it was designed with, but again, you must recreate the report's field structures.

For the most flexible reports, avoid using debase-specific SQL.

Do not terminate SQL statements with a semicolon (;).

WHERE Control

Using a WHERE control lets you simplify WHERE clause constraints in a SELECT control, where they might cause problems. It executes after a record is returned from the database. WHERE controls are only useful when you use them with SELECT controls.

WHERE controls move processing from the database to the client. This adjustment saves database resources and helps you ensure that your SQL is ANSI-standard and thus, portable.

For example, you can select all names from the *staff* table in which the third letter is an "I" — Smith, Thistle, Flint with a SELECT and WHERE clause.

The report block can achieve the same results, and at the same time maintain database portability and move the overhead of record-validation to the calling report process:

```
SELECT name
   FROM staff
```

And using a WHERE control:

```
(substr(block.name, 3, 1) == "I")
```

Each row returned by the SELECT statement loads the current value of the *name* column into the field structure variable *block.name*. Even though *block.name* contains the current value, the returned row is not processed until the WHERE control passes it.

You have a few basic rules to follow when you use WHERE controls:

- The expression must evaluate to a number *or* be capable of being converted to a number.
- Variables (for example, *var =var;* or even *i++)* are not legal in the WHERE clause. You must call a function to perform the assignment internally.
- You may not assign variables. You can call a function to perform an assignment internally.
- Semicolons (;) are illegal.

Trigger Control

Triggers are essentially instructions that control the flow of blocks and their elements. While text areas indicate *what* to write, triggers provide the instructions on how to get the information to write, how to process it, and what to do next.

Showing Other Information

The block detail areas of your report contain primary information. However, you can put supporting, or ancillary data in text areas of headers and footers (pagetext, block, or break). You can have a variable or function that is evaluated and place the returned value in the text. You can put any ascii character in the text areas except the ampersand (&), which signals that the item following it is a variable or function name.

Variables

The ampersand identifies an embedded variable in text. For example, using the `G.TIME` built-in variable in the following pagetext header design looks like this:

```
==== Example report generated &g.time ====
```

It results in the report pagetext header:

```
==== Example report generated 15-MAY-98 ====
```

Functions

To display time instead of (or in addition to) the date, embed the TRIMpl function `to_char()` with the `G.TIME` variable and include a mask (masks are listed in "***Format Masks***" on page 75):

```
== Example report generated &/to_char(g.time, "HH:MI.SS")/ ==
```

The pagetext header would look like this:

```
== Example report generated 11:14.32 ==
```

You can embed any number of functions or variables in a text area. When you embed a *function*, you need to delimit it with a forward slash (/). Also, note that the embedded function requires no semicolon (;).

If an embedded variable name references a variable that doesn't exist, the validation returns an error indicating that the variable can't be found. If an embedded variable or function returns a string that extends beyond the page width, the string wraps to the beginning of the next line and overwrites any text there.

Fields

A *field* in TRIMreport is basically a point at which to place data in text areas of pages, block headers, footers, details, and break sub-blocks. When fields "execute" (in order of their sequence numbers, 1 through *n*), each field's trigger also executes. In each field, the trigger code is similar to

```
{
field = block_name.column_name;
}
```

When the field trigger code executes, the value of *block_name.column_name* is applied to one of the following formats:

- Field data mask (999999 or A5 or MM-DD, YYYY)
- Justification (left, right, or center)

Then, the value is written to the output buffer.

In the example, *field* can be thought of as the current X-Y position. Writing to *field*, for example, causes data to be written to the output buffer:

```
field=123;
```

You can modify the field data's location on a page, including centering, suppressing, or otherwise altering the field. For examples of these customizations, see "***Fields to Center Headings***" on page 46.

The elements of the field are the mask and the trigger. The following table shows the result of values that are too large to fit into a field as defined by its data mask:

Type	Mask	Value	Displayed Result
CHARACTER	A5	"PACIFIC"	PACIF
NUMERIC	999	12345	***

Triggers

Triggers are logical elements, blocks of code, that contain the instructions for each action in a report. Typically you use them to change a report design's execution flow, send information and report data to a printer or other output device, or create, use, and manipulate variables.

Triggers can be local to a report, to a block, or you can define functions external to the design and call them from code inside the design. The MAIN trigger and user triggers are global to the design and block triggers are local to their owning blocks.

TRIMreport has seven types of block triggers:

- PRE-BLOCK
- POST-BLOCK
- POST-HEADER
- PRE-FOOTER
- PRE-FETCH

- POST-FETCH
- CHILD

As you've already read, report block execution begins at the block header, repeats the block detail for each row retrieved by the SELECT and WHERE controls, and executes the block footer. Trigger controls give you access to blocks of code outside the current report block to execute other processes before, between, and after the report block.

You can use triggers more than once in the same report and reuse triggers from other reports.

To create or edit triggers, execute the trigger action for a report object and open the DEFINE TRIGGER window:

- Main trigger (CHOOSE REPORT)
- Report block (DEFINE REPORTBLOCK)
- Page text (DEFINE PAGETEXT)
- Break (DEFINE BREAK)
- Field (DEFINE FIELD)

MAIN Trigger

The *MAIN trigger*, (CHOOSE REPORT > TRIGGER), primarily serves to call another report block or another block of code. A report can have only one MAIN trigger, which is the first piece of code that is executed when you run a TRIMreport design.

User Triggers

User triggers are global functions. You access them in the CHOOSE REPORTBLOCK window, although they are not associated with a particular report block.

You can pass arguments to a function and refer to them internally (to the function) simply as *parm[0]*, *parm[1]*, *parm[2]*, and so on.

The function can also return a value. For example, a function can determine the number of parameters passed to it via `count()`:

```
{
if (count(parm) == 2) printf("two parameters passed");
printf("parameter 1 = "^^parm[0]);
printf("parameter 2 = "^^parm[1]);
return("this is a test");
}
```

NOTE: *If a function references two parameters, you must pass two parameters or you get an error. If a function returns a value, the function call must be part of an expression or an error is generated.*

Block Execution Summary

When the report calls a block, the block and triggers execute in the following order.

1. Execute PRE-BLOCK trigger.
2. Write header text.
3. Execute header fields.
4. Execute POST-HEADER trigger.
5. If SELECT statement returns a row:
    a. Evaluate WHERE control, if defined. If true,
        i. Execute PRE-FETCH trigger.
        ii. Execute POST-FETCH trigger. If there is a break and the last break field is not the same as the current,
            A. For first break group:
                - Execute PRE-FOOTER.
                - Write break header text.
                - Execute break header fields.
                - Execute POST-HEADER trigger.

                For all other break groups:
                - Execute PRE-FOOTER trigger.
                - Write break footer text.
                - Execute break footer fields.
                - Execute break POST-BLOCK trigger.
                - Execute break PRE-BLOCK trigger.
        iii. Write detail text/execute detail fields.
        iv. Execute CHILD trigger.
    a. Return to SELECT.
5. Execute PRE-FOOTER trigger.
    a. Write footer text.
    a. Execute footer fields.
2. Execute POST-BLOCK trigger.

NOTE: *All triggers are optional. WHERE control is always true for a list-driven SELECT control.*

Break Sub-Block

Break sub-blocks add subheadings and group data in a block's detail area. A single block can have any number of break levels and you can define breaks on a value change in any number of fields. Simply specify the variable names in the DEFINE BREAK window.

Break Sub-Block Header & Footer

A break header appears before the first row of data and for each additional break. The break footer appears after the last record in a break group. You can edit either header or footer by DEFINE BREAK > MODIFY.

Break Control Triggers

Like the regular blocks, sub blocks also have trigger controls. The execution, which occurs from break level 1 to *n*, does *not* include its own detail area. A break sub-block only has four types of triggers:

- ***PRE-BLOCK trigge*r —** Executes for each break that occurs, before any other action in the break sub-block (essentially the detail area). Once the break PRE-BLOCK trigger has completed, the break sub-block header is written to the output buffer.
- ***POST-HEADER trigge*r —** If you have defined one, this trigger executes after the break header is written but before the current block's detail area executes.
- ***PRE-FOOTER trigger* —** If you have defined one, a PRE-FOOTER trigger executes after a block's detail area is written for the last row in a group but before the break sub-block footer.
- ***POST-BLOCK trigger* —** If you have defined one, this triggers is the final event for a break group. It executes after the break sub-block's footer is written.

When you use breaks, you should order data on the break column in list-driven SQL statements and note that the SUMDFLT action in DEFINE BREAKS creates SUM fields in the break footer for each numeric field in the detail area.

Special Break Techniques

Basically, a break separates data when the value of a column changes. Sometimes, however, you may need complex breaks.

Using the following table, for example, you can group the data by creating a break each time the value of the first digit of the *dept#* column changes:

DEPT#	EMPLOYEE	SALARY
10	Bill Jones	10,000
11	Mike Hanlon	13,000
12	Kelly Johnson	11,000
22	Joe Morgan	12,500
22	Aldo Bugnon	13,400
23	Bjorn Washington	10,000
34	Isiah Trooper	17,000
34	Gordon Scout	15,000

The data breaks into three groups and the salaries for each group are totalled:

DEPT#	EMPLOYEE	SALARY
10	Bill Jones	10,000
11	Mike Hanlon	13,000
12	Kelly Johnson	11,000

```
                        -------
                        34,000

22          Joe Morgan         12,500
22          Aldo Bugnon        13,400
23          Bjorn Washington   10,000
                               ------
                               35,900

34          Isiah Trooper      17,000
34          Gordon Scout       15,000
                               ------
                               32,000
```

If you create a break on *dept,* a group forms each time the dept field structure changes value. By adding a `substr()` function, you direct the SELECT to create breaks that group the data when only the first digit changes:

```
SELECT substr(to_char(dept),1,1),dept,name,salary
   FROM staff
   ORDER BY dept
```

However, this is not database-independent SQL; it depends on the database performing necessary hashing functions `to_char()` and `substr()`. For instructions on making a report database-independent, see "***Making a Report Database-Independent***" on page 44.

Chapter 3
Design Considerations

Effective Designs

Designing your report involves making tradeoffs. One important consideration you face is balancing complexity with efficiency and maintainability. A complex design may yield the most informative report, but it may also be more difficult to maintain and take longer to run.

At the same time, a simple report might also be inefficient and slow. For example, if a report design requires database information to be printed out three times in a report and sorted on a different column each time, you can design your report to retrieve data from the database three times, ordered on a different column each time, or you can retrieve the data once, put it in a list, and sort it three times.

Other factors you need to consider when designing your report:

- Distribution of work between local (TRIMpl) and remote processing (database), including database traffic, time requirements, and complexity.
- Design complexity and maintainability, including schedule deadlines, time requirements, and maintenance.

Recommendations:

- The database is a shared resource. Use it sparingly.
- Keep the report as simple as makes sense.

Report Design Methods

You can generate reports in a variety of ways; the most effective and efficient reports usually comprise two or sometimes all three methods available to you with TRIMreport:

- ***SELECT-driven*** — uses subsets of table data. The report writer executes a detail area for each row returned by a SELECT statement.
- ***List-driven*** — creates lists and extracts data from those lists (still uses SELECT).
- ***Trigger-driven*** — calls report blocks. Fields in each block extract information, usually from lists.

Distributing Work

The way you write your queries and reports controls whether data processing is performed on the database machine or on the reportwriter's machine. For example, you may write SQL that prints out the amount saved by applying a 10 percent budget cut across operating expenses for each department. Requesting the processed information from the database generates the new values. For example,

```
SELECT (budget - (budget * 0.10)) FROM table
```

Or you can retrieve the data, store it in a list and process it locally:

```
SELECT budget FROM table
```

The second approach requires more complex processing but uses fewer database resources since the database does not have to retrieve and process all the data rows. If many people are using the database at the same time, the first approach has a greater impact on other users than the second option.

You can also control the amount of work you require from the database by the way you use TRIMreport commands. The same principles apply here: the simple version has a straightfoward design and fast development time, but it's larger and requires more from the database.

For example, if you need to display data sorted three different ways, you could create a TRIM block report:

```
BLOCK 1: SELECT * FROM table ORDER BY column1
BLOCK 2: SELECT * FROM table ORDER BY column2
BLOCK 3: SELECT * FROM table ORDER BY column3
```

The single-block report design option, which eliminates redundant code and reduces the database interaction to one query is more memory-intensive at runtime and increases the complexity and report-writing maintenance:

```
List = SELECT * FROM table ORDER BY column1
Call ALL_PURPOSE_BLOCK
SORT list by column 2
Call ALL_PURPOSE_BLOCK
SORT list by column 3
Call ALL_PURPOSE_BLOCK
```

SELECT-driven Method

The SELECT-driven method produces reports that are subsets of table data (and thus produce meaningful information for the breaks, break SUM (group), and SUM (total) defaults). The detail area is executed for each row returned from the database by the SELECT statement. The SELECT-driven method relies on the SELECT control.

The SELECT method is not dynamic. For example, a report designed to retrieve specific columns from a database table or tables expects the exact tables to exist whenever the report is executed.

Default Behavior

You can instruct TRIMreport to generate default code for report designs with CREATE:

When you CREATE…	TRIMreport generates…
Report	Main trigger.
First block	Block call in the main trigger.
Child block	Child trigger in the parent block.
Sibling block	POST-BLOCK trigger in the last block in the sibling chain.

When you create a SELECT control, you have several options for laying out the data:

- DEF_HORZ creates fields in the detail area that correspond to the columns in the SELECT.
- DEF_VERT lays the fields out top down without text.
- DEF_VER2 results in top-down field layout with column names the left of the fields.

The fields are automatically given triggers, such as the one below, and text is added to the blocks' header text.

```
{
field = block-name.column-name;
}
```

In default behavior the main trigger calls the first block and the first block uses its CHILD trigger to call a child block for each row returned. The child block executes its SELECT and calls its sibling block via the POST-BLOCK trigger. For example, you can lay out the report:

```
Main trigger calls BLK_A: block(BLK_A);
    Execute BLK_A's SELECT
        For each row returned, execute the CHILD
            trigger block (BLK_z);
    Execute BLK_z's SELECT
        After last row, execute POST-BLOCK trigger:
            block(BLK_y);
    Execute BLKY_y's SELECT
        After last row, execute POST-BLOCK trigger:
            block(BLK_B);
    Execute BLK_B's SELECT
        After last row, execute POST-BLOCK trigger:
        block(BLK_C);
    Execute BLK_C's SELECT
        After last row, execute POST-BLOCK trigger:
            block(BLK_t);
    Execute BLK_t's SELECT
```

The example that follows creates a SELECT-driven report that groups sales personnel data by division. The main trigger prints a page header and calls the main report block and prints the page footer. The main report block specifies the header text including the `G.PAGENUMBER` variable. The SELECT statement is executed for the header

```
SELECT * FROM org
```

The main report block detail is printed and the data and attributes are defined for each field. For example,

```
Field: Division
Row offest: 3, Column offset: 22, Field width: 10
Mask: A10
Trigger: FIELD
    Text: Trigger/Function
        1> {
        2> field = MAIN.DIVISION;
        3> }
```

A blank report block footer is specified and control passes to the CHILD trigger which calls the child report block (MAIN_1). The SELECT statement for the child report block is executed:

```
SELECT id, name, job, salary, dept FROM staff WHERE
             DEPT = &MAIN.DEPTNUMB
```

The child report block header for the group prints, followed by the detail lines for each member of the group. When the last row of data for the group is complete, the child report block footer prints. The footer calculates and prints the total of salaries for the group — `sum(SALARY)`.

Control returns to the MAIN report block to begin processing the next group. Each time the SELECT is processed, data is retrieved directly from the database.

Examples

This SELECT-driven report design printed below was generated with TRIMlis, a documentation utility. The syntax for generating a listing of the report design is

```
trimlis report-file-name
```

For more information on the TRIMlis utility, refer to the *DesignVision User Guide.*

Report Design

```
DESIGN REPORT ROOT NODE Version: 2216
   Name: SALES_SALARIES
   Creator:  L.L. Langtry        Create date: 28-JAN-1992 at 08:01
   Modifier: L.L. Langtry        Modify date: 28-JAN-1992 at 13:01
   Page length: 60, Page width: 80

Trigger: MAIN
   Text: Trigger/Function
     1>{
     2>open();                   /* open output file              */
     3>pageinate(header);       /* print pageheader              */
     4>block(MAIN);             /* first block                   */
     5>paginate(footer|break); /* print page footer and break   */
     6>close ();                 /* close output file             */
     7>}

DESIGN REPORTBLOCK
Name: MAIN
   Page:
   Page length: 60, Page width: 80
   Text: Report block select statement
     1>SELECT * FROM org

   Text: Report block header
     1>
     2>                            SALES STAFF SALARIES
     3>                                   (by Division)
     4>                                           Page
     5>
```

```
  Field: PgNo
  Row offset: 3, Column offset: 69, Field width: 3
  Mask: 999
  Trigger: FIELD
    Text: Trigger/Function
        1>field = G.PAGENUMBER;

Text: Report block detail
  1>
  2> Department Number:                            Name:
  3>           Manager:
  4>          Division:                          Location:
  5>
  Field: DeptNumb
  Row offset: 1, Column offset, 22, Field width: 3
  Mask: 999
  Trigger: FIELD
    Text: Trigger/Function
        1>{
        2>field = MAIN.DEPTNUMB;
        3>}

  Field: Manager
  Row offset: 2, Column offset, 22, Field width: 3
  Mask: 999
  Trigger: FIELD
    Text: Trigger/Function
        1>{
        2> FIELD = MAIN.MANAGER;
        3>}

  Field: Division
  Row offset: 3, Column offset: 22, Field width: 10
  Mask: A10
  Trigger: FIELD
    Text: Trigger/Function
        1>{
        2> field = MAIN.DIVISION;
        3>}

  Field: DeptName
  Row offset: 1, Column offset: 55, Field width: 16
  Mask: A16
  Trigger: FIELD
    Text: Trigger/Function
        1>{
        2> field = MAIN.DEPTNUM;
        3>}

  Field: Location
  Row offset: 3, Column offset: 55, Field width: 14
  Mask: A16
  Trigger: FIELD
    Text: Trigger/Function
        1>{
        2> field = MAIN.LOCATION;
```

```
              3>}

            Text: Report block footer
              1>
              2>

          Trigger: CHILD
            Text: Trigger/Function
              1>block(MAIN_1);

Child reportblock:
DESIGN REPORTBLOCK
Name: MAIN_1
    Text: Report block select statement
      1>SELECT id,name,job,salary,dept FROM staff
    Text: Report block where filter
      1>(DEPT == MAIN.DEPTNUMB)

    Text: Report block header
      1>            ID        NAME         JOB        SALARY
      2>          ------   -------------   -----       --------

    Text: Report block detail
      1>

      Field: ID
      Row offset: 0, Column offset: 11, Field width: 3
      Mask: 999
      Trigger: FIELD
        Text: Trigger/Function
            1>{
            2> field = MAIN_1.ID;          /* Move data to field    */
            3>}

      Field: NAME
      Row offset: 0, Column offset: 19, Field width: 12
      Mask: A12
      Trigger: FIELD
        Text: Trigger/Function
            1>{
            2> field = MAIN_1.NAME;        /* Move data to field    */
            3>}

      Field: JOB
      Row offset: 0, Column offset: 39, Field width: 5
      Mask: A5
      Trigger: FIELD
        Text: Trigger/Function
            1>{
            2> field = MAIN_1.JOB;        /* Move data to field    */
            3>}

      Field: SALARY
      Row offset: 0, Column offset: 52, Field width: 10
      Mask: $999999.99
      Trigger: FIELD
        Text: Trigger/Function
```

```
          1>{
          2> field = MAIN_1.SALARY;     /* Move data to field    */
          3>}

    Text: Report block footer
      1>
      2>
      3>

      Field: SUM_SALARY
      Row offset: 1, Column offset, 52, Field width: 10
      Mask: $999999.99
      Trigger: FIELD
        Text: Trigger/Function
          1>field = sum(SALARY);
          3>}
```

Formatted Report

```
                SALES STAFF SALARIES
                   (by Division)
                                                       Page 1
Department Number: 10                             Name: HEAD OFFICE
          Manager: 160
         Division: CORPORATE                  Location: NEW YORK

       ID        NAME      JO     SALARY
       ----     --------- ----------------
       25        MOLINARE  MGR    $22959.20
       25        LU        MGR    $20010.00
       25        DANIELS   MGR    $19260.25
       25        JONES     MGR    $21234.00
                                  ===========
                                  $83463.45

Department Number: 15                             Name: NEW ENGLAND
         Manager: 50
       Division: EASTERN                    Location: NEW YORK

       ID        NAME      JO     SALARY
       ----     ---------  -----  -----------
       25        HANES     CLERK  $20659.80
       25        ROTHMAN   CLERK  $16502.83
       25        NGAN      CLERK  $12508.20
       25        KERMISCH  CLERK  $12258.5
                                  ===========
                                  $61929.33
```

List-Driven Method

One reason TRIMreport is so flexible and powerful is because of its *list* feature. A list in TRIM terms is a matrix of data stored as a variable. Each cell in the list can contain data of any type, including another list. You can manipulate the columns of data in the lists with the array of built-in list functions.

In a list-driven report design, data for the report is retrieved from the database and stored in a list. Data manipulation required by the report can then be performed locally on the list. Report blocks in the design can select all or portions of the data from the list for the report.

In the example that follows, a report list is created with `list_open()`. This function can be used to define a list, load a list from a file, or load a list from the database. In a list-driven report design, a list variable is used in place of a SELECT statement with the list data described in a SELECT:

```
ol = list_open("SELECT deptnumb, deptname, manager
                   FROM org", 100);
```

The variable declaration defines the list, *ol*, as having up to 100 rows of data retrieved from three columns by a SELECT statement.

The data in the list can be selected by a list reference in the report block SELECT control:

```
[g.ol, deptnumb, deptname, manager]
```

The first time the block is called, the contents of each column of the list *g.ol* is copied to the specified variables, deptnumb, dept name, and manager. After the detail area is placed, the current row pointer increments and the values copied out again. The detail area is repeated for each row starting from the current row to the last row (100 or less) in the list.

To write out the data, you can create fields and include specifications such as justification and data type:

```
{
field = MAIN.DEPTNAME;
}
```

Examples

Report Design

```
DESIGN REPORT ROOT NODE Version: 2216
    Name: trig_d
    Creator:  J. Smith        Create date: 10-FEB-1992 at 09:01
    Modifier: J. Smith        Modify date: 12-FEB-1992 at 12:01
    Page length: 56, Page width: 80

Trigger: MAIN
    Text: Trigger/Function
      1>{
      2>list ol;
      3>list sl;
      4>
      5>ol = list_open("SELECT deptnumb,deptname,manager FROM org",100);
      6>open();                           /* open output file    */
      7>paginate(header);                 /* print page header   */
      8>block(MAIN);                      /* first block         */
      9>paginate(footer|break);   /* print page footer and break */
     10>close();
     11>}
```

```
DESIGN REPORTBLOCK
Name: MAIN
    Page:
    Page length: 56, Page width: 80
    Text: Report block select statement
      1>[g.ol,deptnumb,deptname,manager]

    Text: Report block detail
      1>
      2> Department ID:                    Department Name:
      3>       Manager:
      4> =====================================================
      5>
      Field: DEPTNAME
      Row offset: 1, Column offset, 52, Field width: 16
      Mask: A16
      Trigger: FIELD
        Text: Trigger/Function
            1>{
            2>field = MAIN.DEPTNAME;     /* Move data to field     */
            3>}

      Field: MANAGER
      Row offset: 2, Column offset, 17, Field width: 3
      Mask: 999
      Trigger: FIELD
        Text: Trigger/Function
            1>{
            2> field= MAIN.MANAGER;
            3>}

      Field: DEPTNUMB
      Row offset: 2, Column offset, 17, Field width: 3
      Mask: 999
      Trigger: FIELD
        Text: Trigger/Function
            1>{
            2> field = MAIN.DEPTNUMB;
            3>}

      Trigger: CHILD
        Text: Trigger/Function
            1>{
            2>g.sl = list_open("SELECT id,name,salary FROM staff"
            3>                   "WHERE dept = &main.deptnumb ",100);
            4>block(MAIN_1);
            5>}

Child reportblock:
DESIGN REPORTBLOCK
Name: MAIN_1
    Text: Report block select statement
      1>[g.sl,id,name,salary]

    Text: Report block detail
      1>
```

```
Field: ID
Row offset: 0, Column offset: 2, Field width: 8
Mask: 99999999
Trigger: FIELD
  Text: Trigger/Function
      1>{
      2> field = MAIN_1.ID;          /* Move data to field    */
      3>}

Field: SALARY
Row offset: 0, Column offset: 55, Field width: 8
Mask: 999999.99
Trigger: FIELD
  Text: Trigger/Function
      1>{
      2> field = MAIN_1.SALARY;      /* Move data to field    */
      3>}

Field: NAME
Row offset: 0, Column offset: 24, Field width: 10
Mask: A10
Trigger: FIELD
  Text: Trigger/Function
      1>{
      2> field = MAIN_1.NAME;        /* Move data to field    */
      3>}
```

Formatted Report

The top of the first page of the formatted report looks like this:

```
  Department No:  10             Department Name: HEAD OFFICE
        Manager: 160

=================================================================
    160             MOLINARE  22959.20

    210             LU        20010.00

    240             DANIELS   19260.25

    260             JONES     21234.00


  Department No: 15              Department Name: NEW ENGLAND
        Manager: 50

=================================================================
     50             HANES     20659.80

     70             ROTHMAN   16502.83

    110             NGAN      12508.20

    170             KERMISCH  12258.50
```

```
 Department No: 20             Department Name: MID ATLANTIC
      Manager: 10

==============================================================
    10            SANDERS   18357.50

    20            PERNAL    18171.25

    80            JAMES     13504.60

   190            SNEIDER   14252075
```

Trigger-Driven Method

The trigger-driven method of report design uses triggers to call report blocks and place headers or footers, forcing them to take the place of block detail.

Trigger-driven reports are completely dynamic, parameter-driven reports that allow you to bypass most of TRIMreport's implicit controls.

Generally, the control for the trigger-driven method is the main trigger.

The example that follows prompts for a table name, retrieves data from the table into a list, *ll*, and uses the MAIN trigger to call a report block, PRINT_ROW.

```
{
list ll;
char table[30];

table = prompt("Please enter TABLE name>");
ll = list _open("SELECT * FROM "^^table,100);
open();
pageinate(header);
while (list_rows(ll)) {           /* While there are rows     */
   block(PRINT_ROW);              /* Print a row              */
   list_mod(ll,0);                /* Delete a row             */
   }
paginate(footer | break);
close();
}
```

The block called by the MAIN trigger, PRINT_ROW, contains only a header area with one field of character data that is right-justified and has an A5 mask.

```
{
int i;
for (i = 0; i<list_col(g.ll); i++){          /* For each col     */
   field = list_curr (g.ll,i);               /* Print col value  */
   g.pageoffset = g.pageoffset + 10;         /* Move 10 spaces   */
   }
g.pageoffset = 0;                            /* Reset to zero    */
```

The example prompts the user for a table name, then dynamically selects rows from the table and prints/displays all the columns in the table. Trigger-driven reports preceded the list-driven alternative and remains useful if you need to create completely dynamic reports.

Examples

Report Design

In the example that follows the main trigger retrieves data into two lists, *ol* and *ml***.** First data is retrieved for the *ol* list

```
SELECT deptnumb, deptname, manager FROM org
```

Then data is retrieved for the *ml* list with a WHERE condition based on values in the *ol* list:

```
SELECT id, name,dept,salary FROM staff
WHERE dept = "^^list_curr(ol,0)^^"
```

The SELECT statement imposes a limit of 100 on the number of rows retrieved into the list. This is sufficient for the entire *org* and *staff* example database tables. If the tables contained more than 100 rows, the `list_more()` function can retrieve additional data from the database.

The MAIN report block contains the header and defines the fields — *m1, m2, m3* — into which data is retrieved.

The first sibling report block, STAFF, defines the three fields — *o1, o2, o3* — into which the data is placed.

The second sibling report block, SA, specifies a header and field — *sal* — into which the salary total for each group is placed.

```
DESIGN REPORT ROOT NODE Version: 2216
    Name: trig_d
    Creator:  J. Smith        Create date: 10-FEB-1992 at 09:01
    Modifier: J. Smith        Modify date: 12-FEB-1992 at 12:01
    Page length: 56, Page width: 80

Trigger: MAIN
    Text: Trigger/Function
      1>{
      2>list ml;
      3>list ol;
      4>int i,j;
      5>numeric tsalary;
      6>
      7>open();                          /* open output file    */
      8>paginate(header);                /* print page header   */
      9>
     10>ol=list_open("SELECT deptnumb,deptname,manager FROM org",100);
     11>for(i;0; i < list_rows(ol); i++ {
     12>block(MAIN);                     /* first block         */
     13>ml = list_open("SELECT id,name,dept,salary FROM staff"
     14>               "WHERE dept = "^^list_curr(ol,0)^^" ",100);
     15> tsalary = 0;
     16> for (j=0; j < list)rows (ml); j++ {
     17>   block(STAFF);
     18>  tsalary = tsalary +list_curr(ml,3);
     19>  list_next(ml);
     20>  }
     21> block(SA);
     22> list_next(ol);
     23> }
     24>paginate(footer|break);   /* print page footer and break */
     25>close();
     26>}

DESIGN REPORTBLOCK
Name: MAIN
    Page:
    Page length: 56, Page width: 80
    Text: Report block header
      1>
      2>
      3>   Department No:           Department Name:
      4>   Manager:
      5>   =======================================================
      Field: m1
      Row offset: 2, Column offset: 17, Field width: 2
      Mask: 99
      Trigger: FIELD
        Text: Trigger/Function
            1>field = list_curr(g.ol,0);

      Field: m2
      Row offset: 2, Column offset: 47, Field width: 20
      Mask: A20
      Trigger: FIELD
        Text: Trigger/Function
            1>field = list_curr(g.ol,1);

      Field: m3
      Row offset: 3, Column offset: 11, Field width: 20
      Mask: A20
      Trigger: FIELD
        Text: Trigger/Function
            1>field = list_curr(g.ol,2);

Sibling reportblock:
DESIGN REPORTBLOCK
Name: STAFF
    Page:
    Page length: 56, Page width: 80
    Text: Report block header
      1>
      2>
      Field: o1
      Row offset: 0, Column offset: 3, Field width: 3
      Mask: 999
      Trigger: FIELD
        Text: Trigger/Function
```

```
          1>field = list_curr(g.ml,0);

     Field: o2
     Row offset: 0, Column offset: 20, Field width: 20
     Mask: A20
     Trigger: FIELD
       Text: Trigger/Function
          1>field = list_curr(g.ml,1);

     Field: o3
     Row offset: 0, Column offset: 45, Field width: 10
     Mask: 999999.99
     Trigger: FIELD
       Text: Trigger/Function
          1>field = list_curr(g.ml,3);

Sibling reportblock:
DESIGN REPORTBLOCK
Name: SA
   Page:
   Page length: 56, Page width: 80
   Text: Report block header
     1>
     2>

   Field: sal
   Row offset: 0, Column offset: 45, Field width: 10
   Mask: 999999.99
   Trigger: FIELD
       Text: Trigger/Function
          1>field = g.tsalary;
```

Formatted Report

The top of the first page of the formatted report looks like:

```
 Department No: 10             Department Name: HEAD OFFICE
 Manager: 160

=================================================================
   160             MOLINARE  22959.20

   210             LU        20010.00

   240             DANIELS   19260.25

   260             JONES     21234.00

                            Total salary =83463.45
 Department No: 15             Department Name: NEW ENGLAND
 Manager: 50

   =================================================================
    50             HANES     20659.80

    70             ROTHMAN   16502.83
```

```
  110             NGAN      12508.20

  170             KERMISCH  12258.50

                     Total salary =61929.33

Department No: 20             Department Name: MID ATLANTIC
Manager: 10

=================================================================
   10             SANDERS   18357.50

   20             PERNAL    18171.25

   80             JAMES     13504.60

  190             SNEIDER   14252075

                   Total salary =64286.10
```

Chapter 4
Designing a Report

When you begin working with TRIMreport, you probably want to have the application create default designs for you to work with. To develop your own design, you modify elements in the default, and where necessary, add new elements. This chapter describes typical tasks of most report development and walks you through the procedures.

Step by Step

The first task, once TRIMreport is installed, is to start the application. To get defaults, you must connect to a database and identify tables and columns for your report.

1. Start TRIMreport and connect to a database following the instructions for your operating system and database in the Connect Guide.

   For example:

   ```
   trimrep uid/pwd [Enter]
   ```

NOTE: *As you complete this introduction, read each procedure carefully before executing any commands. TRIMreport has NO "undo" button. If you make a mistake and cannot remember how to "step backwards" you'll have to start from the beginning.*

☞ *Creating a basic report*

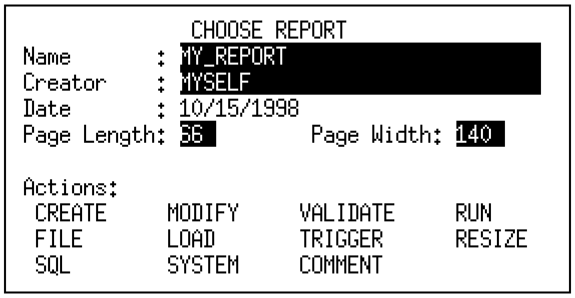
You construct reports using blocks. Your entire report can be in a single block or many blocks. You create and maintain each block individually. This introduction uses a table common to many databases, STAFF, and creates a single-block report.

The first window that appears when you start TRIMreport is the `CHOOSE REPORT`. To create your report, begin by naming the report, identifying the creator (you), and specifying a page length and width, as illustrated. [Tab] from one field to the next. Then create a single report block with the same name as the report. For reports with multiple blocks you can use any names that are meaningful.

NOTE: *If you are running against an Oracle database, specify 140 for the page width. For all other databases, 80 characters is wide enough to fit all the columns. (Oracle leaves lots of extra room for numbers and thus, the columns are wider than for any other database.)*

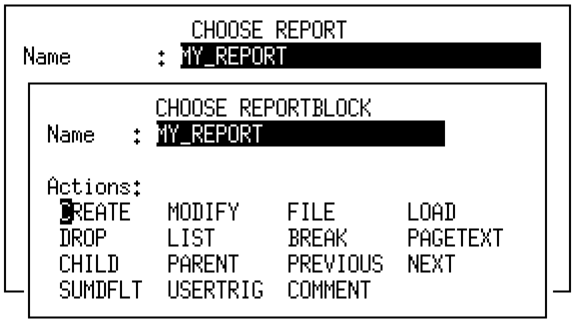
1. Type "**`MY_REPORT`**", press [Tab]. (Note: Must use an underscore.)
2. Type "**`MYSELF`**" (or your name, if you prefer) and press [Tab].
3. Type "**`66`**", press [Tab], type "**`80`**"if you're not using an Oracle database, "**`140`**" if you are.

Your screen should look like this:



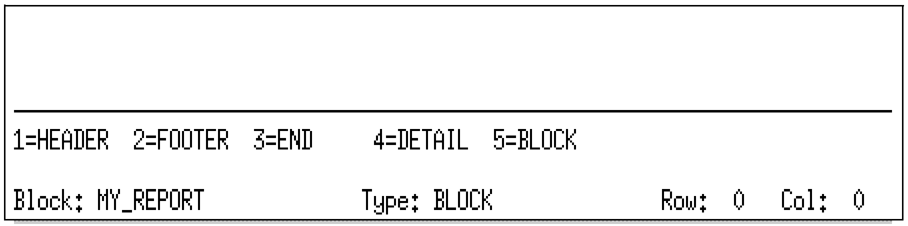
```
                CHOOSE REPORT
Name       : MY_REPORT
Creator    : MYSELF
Date       : 10/15/1998
Page Length: 66          Page Width: 140

Actions:
 CREATE     MODIFY     VALIDATE     RUN
 FILE       LOAD       TRIGGER      RESIZE
 SQL        SYSTEM     COMMENT
```

4. [Tab] to the `CREATE` action and press [Enter].
5. Type "**`MY_REPORT`**" and [Tab] to the `CREATE` action and press [Enter].



```
                CHOOSE REPORT
Name       : MY_REPORT

             CHOOSE REPORTBLOCK
  Name   : MY_REPORT

  Actions:
   CREATE   MODIFY    FILE      LOAD
   DROP     LIST      BREAK     PAGETEXT
   CHILD    PARENT    PREVIOUS  NEXT
   SUMDFLT  USERTRIG  COMMENT
```

You'll see a blank screen that represents the empty report detail and a row of active function keys at the bottom:



```
1=HEADER  2=FOOTER  3=END     4=DETAIL  5=BLOCK

Block: MY_REPORT            Type: BLOCK            Row:  0   Col:  0
```

☞ *Writing the Block Control*

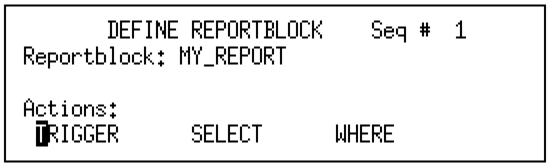
Now you define the report block, that is specify this block's data-retrieval instructions. In this case, you define a SELECT control with a single SELECT statement.Type the SELECT statement as it appears in the following dialog box. This is standard SQL, which you should recognize.

The asterisk (*) specifies all the columns in the table and ORDER BY specifies the columns on which to arrange the data numerically. Note that unlike standard SQL, however, you do ***not*** end the line with a semicolon(;).

After you complete the SQL, you can validate or file the statement, load another statement that was previously saved, and specify the report to be horizontal or vertical. When you choose a report style and press [Enter], TRIMreport accesses the database, selects the definitions and labels of all the specified columns in the specified tables and returns an "OK" at the bottom of the screen when the query is complete.

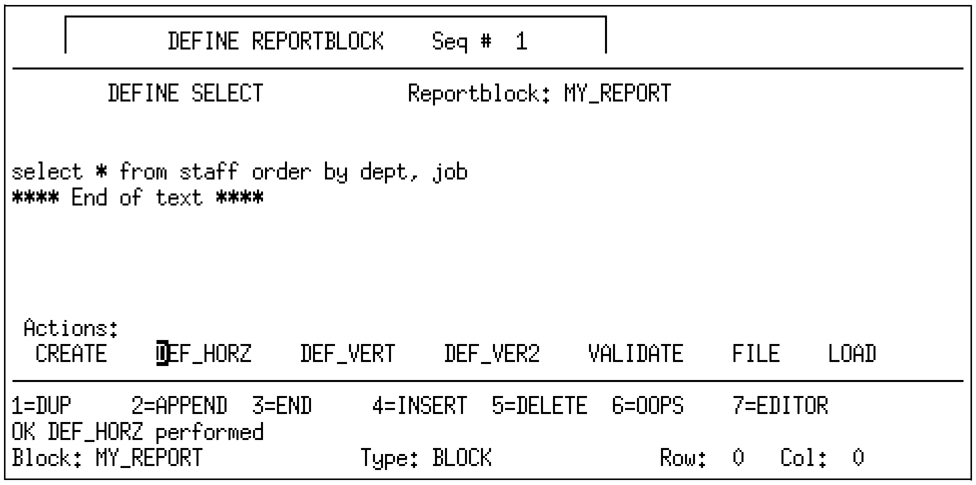
NOTE: *SQL is conventionally written in uppercase, but case only matters if you are working with a Sybase database. Since Sybase databases distinguish between upper and lower case, you must know the exact representation of* ***table*** *and* ***column*** *names.*

1. Press [F5] to display the `DEFINE REPORTBLOCK` dialog box.



```
      DEFINE REPORTBLOCK    Seq #  1
Reportblock: MY_REPORT

Actions:
 TRIGGER      SELECT      WHERE
```

2. [Tab] to the `SELECT` action and press [Enter].
3. In the `DEFINE SELECT` dialog that appears on your screen, make sure that your cursor is on the line `**** End of text ****`.
4. Press [F4] to open a line for typing.
5. Type "**`select * from staff order by dept, job`**"



```
          DEFINE REPORTBLOCK    Seq #  1

       DEFINE SELECT           Reportblock: MY_REPORT


select * from staff order by dept, job
**** End of text ****




 Actions:
  CREATE    DEF_HORZ    DEF_VERT    DEF_VER2    VALIDATE    FILE    LOAD

1=DUP     2=APPEND  3=END     4=INSERT  5=DELETE  6=OOPS    7=EDITOR
OK DEF_HORZ performed
Block: MY_REPORT             Type: BLOCK             Row:  0   Col:  0
```

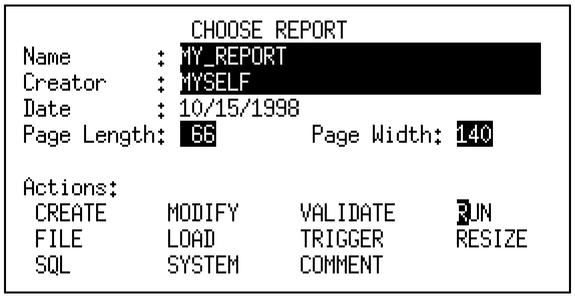
6. [Tab] to `DEF_HORZ` and press [Enter].
7. After you see the status "`OK DEF_HORZ performed`" at the bottom of the window, press [F3] twice to see the default report definitions.

If you didn't specify enough characters to fit all the detail on one line, it wraps. If the page width is wider than the window, you can scroll to the right and left to see all parts.

☞ *Running the report*

Running the report actually instructs TRIMreport to first validate the design and then select the specified data (from the SELECT control) from the database to create your report.

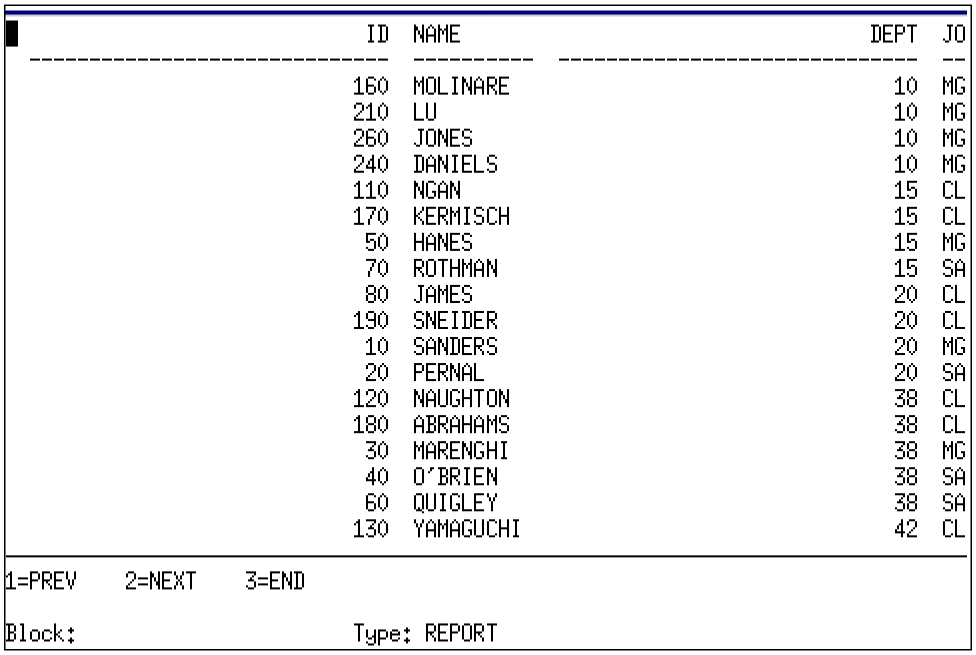
1. Press [F3] twice to go to the `CHOOSE REPORT` dialog.



```
              CHOOSE REPORT
Name        : MY_REPORT
Creator     : MYSELF
Date        : 10/15/1998
Page Length:  66        Page Width: 140

Actions:
 CREATE     MODIFY     VALIDATE     RUN
 FILE       LOAD       TRIGGER      RESIZE
 SQL        SYSTEM     COMMENT
```

2. [Tab] to the `RUN` action and press [Enter].

This is what appears in the report:



```
                              ID  NAME                                DEPT  JO
 ------------------------------  ----------  ------------------------------  --
                             160  MOLINARE                              10  MG
                             210  LU                                    10  MG
                             260  JONES                                 10  MG
                             240  DANIELS                               10  MG
                             110  NGAN                                  15  CL
                             170  KERMISCH                              15  CL
                              50  HANES                                 15  MG
                              70  ROTHMAN                               15  SA
                              80  JAMES                                 20  CL
                             190  SNEIDER                               20  CL
                              10  SANDERS                               20  MG
                              20  PERNAL                                20  SA
                             120  NAUGHTON                              38  CL
                             180  ABRAHAMS                              38  CL
                              30  MARENGHI                              38  MG
                              40  O'BRIEN                               38  SA
                              60  QUIGLEY                               38  SA
                             130  YAMAGUCHI                             42  CL

1=PREV    2=NEXT    3=END

Block:                        Type: REPORT
```

Saving a Report's Format

While it does look like all the information is in the report (the columns extend farther right than the window is wide), this isn't a report you could hand to your manager. Now it's time to improve the report's appearance, changing the data format, adding breaks, and a report title. But first, save the current format.

1. Press [F3] to return to the `CHOOSE REPORT` dialog box.
2. [Tab] to the `FILE` action and press [Enter].
3. Accept the default name and press [Enter] to file (save) the design in your local directory.

   The status line reads `OK FILE performed`

Modifying the Report

You modify the report's design, or format, in the report design window, which shows the default header and field detail lines. You are going to remove the ID column and heading, move a couple columns and their headers, rename some headers, change the format masks here and there, and add some symbols to clarify values. To access each of the report areas (header, footer, and detail) press the associated function key.

1. On the `CHOOSE REPORT` dialog, [Tab] to the `MODIFY` action and press [Enter].
2. On the `CHOOSE REPORTBLOCK` dialog, [Tab] to the `MODIFY` action, and press [Enter].

Remove Column & Header

1. Press [F4] to enter the detail area.
2. Put your cursor in the first series of 9s and press [F4] to cut the detail.

   If you want to make sure it's the correct column first, press [F2] and make sure that `Name: ID` before you press [F4] to cut.
3. To remove the header text, press [F3] to leave detail, [F1] to enter Header and [F1] again to edit text.
4. Type over the line and the label "ID" putting blank spaces in with the spacebar.

NOTE: *Be really careful here, because there's no undo or backup, unless that has been mapped. You must know how your keys have been mapped.*

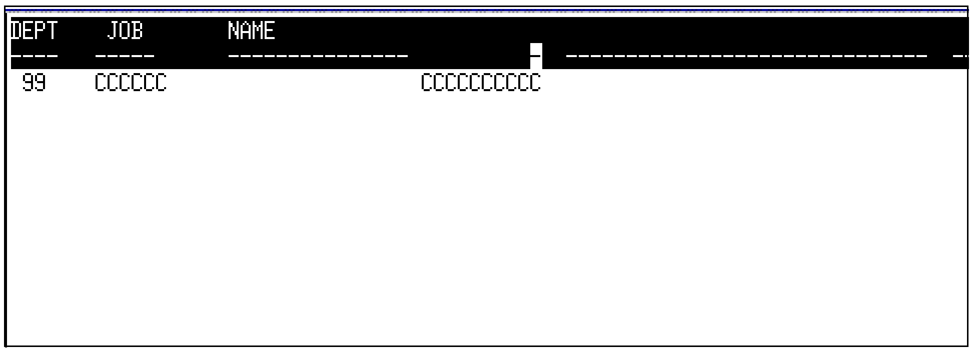
Adjust Format Mask

Before you move the DEPT column and heading, adjust the format mask. DEPT only needs two digits.

1. Press [F3] to leave the header area, [F4] to edit detail, and [Tab] to the 999 under DEPT.
2. Press [F2] for the `DEFINE FIELD` dialog.
3. [Tab] to the `Mask` field and type 99. Press the spacebar to remove the rest of the 9s.
4. Press [F3] to return to the detail area.

Move Columns & Headers

1. Press [F4] to cut the detail, move the cursor to the right edge of your screen, and press [F5] to paste the detail in its new location.
2. Move the detail for "JOB" the same way. [Tab] to the Cs under or to the left of "JO" (JOB may be truncated since heading information is screen-dependent), press [F4], move your cursor to the new location (about 5 spaces to the right of the DEPT's 99) and press [F5].
3. Now put new labels in the header. Press [F3] to leave the detail area and [F1] twice to enter the header text area.
4. Type **`DEPT`** and **`JOB`** with a row of dashes over the detail items. While we're modifying the header text, retype **`NAME`** about five spaces away from the end of the JOB detail, and put a dashed line under it, as illustrated below.



```
DEPT    JOB       NAME
----   -----      ---------------          _  -----------------------------  -
 99    CCCCCC                     CCCCCCCCCC
```

5. To move the detail for NAME, return to the detail area ([F3], [F3], [F4]) and shift the column to the left (put your cursor in the new beginning location and press [F6]).

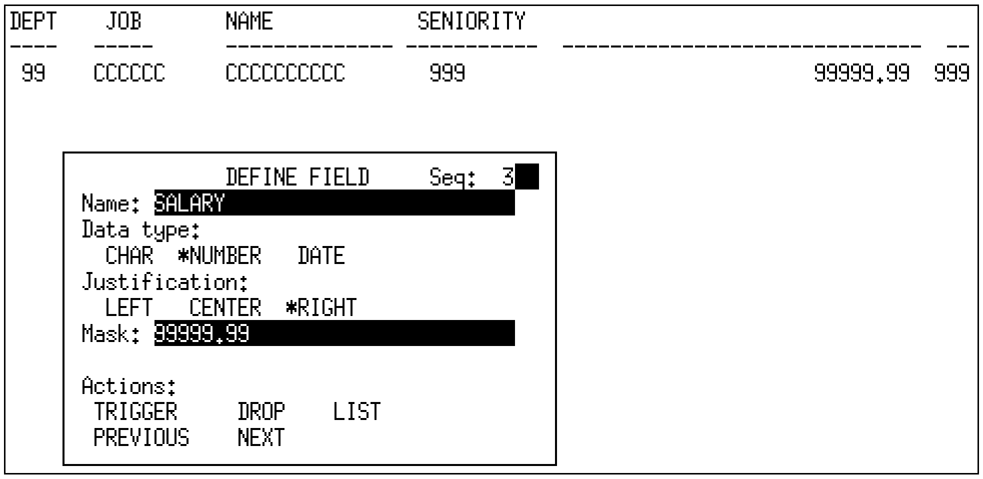
   If your report is wider than your screen, you see the 999s of a new column appear to the right.

6. Move the cursor and continue to shift the new column left [F6] until it's about 5 spaces away from the end of the NAME column.

Change Masks

The first label is YEARS, which refers to the number of years the employee has been at the company. Change the mask so there are only three digits, which is more logical for that information. If you can remember all the detail items, you can edit and adjust them at one time and then go into the header area and make your changes there. If this is your first time using a reportwriter, you may want to take the extra time to switch between the detail fields and the header text.

1. Put your cursor on the first 9 and press [F2].
2. [Tab] to the Mask field, type 999 and press the spacebar to erase the rest of the numbers. Press[F3] when you're finished.
3. Press [F3], [F1], and [F1] to go to the header text area. Type "**`SENIORITY`**" over the 999 you just edited and center it over the numbers.
4. Press [F3] twice, [F4], [Tab] to the detail, and press [F2] to examine the next detail column.



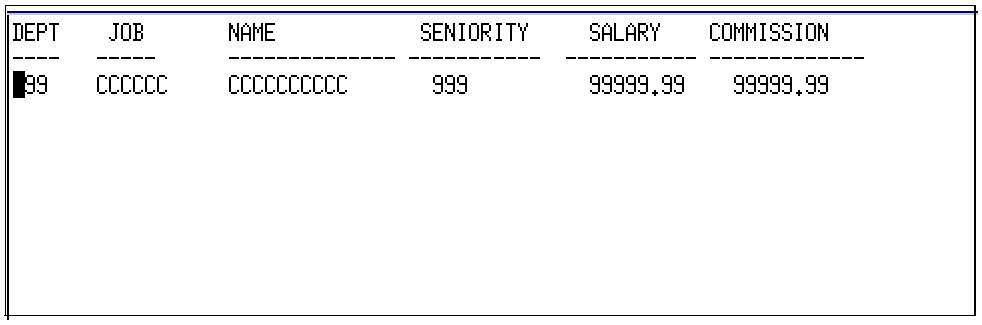
```
DEPT    JOB       NAME           SENIORITY
----   -----     ------------- -----------  ----------------------------  --
 99    CCCCCC    CCCCCCCCCC       999                           99999.99  999


            DEFINE FIELD     Seq:  3
   Name: SALARY
   Data type:
     CHAR  *NUMBER   DATE
   Justification:
     LEFT   CENTER  *RIGHT
   Mask: 99999.99

   Actions:
    TRIGGER     DROP    LIST
    PREVIOUS    NEXT
```

The mask for this field is correct, so adjust the column to the left a bit ([F3], move the cursor to the left and [F6] until you are satisfied with the placement) and put a label in the header area.

([F3], [F1], [F1] and type)

Repeat your actions for the next (last) unidentified field, which turns out to be labelled COMM. This field shows the commissions for each employee. Move the field and enter a suitably descriptive label for the column. Clean up any leftover header text. Make sure you scroll all the way to the right to find any text there, especially if you are using an Oracle database.

Your report design area should look like this:



```
DEPT    JOB       NAME           SENIORITY     SALARY    COMMISSION
----   -----      -------------- -----------  ----------- -------------
999    CCCCCC     CCCCCCCCCC       999          99999.99    99999.99
```

Resize

If you created the report with a width of 140, you can resize the "page" and recenter the headings.

1. Press [F3] until you get to the `CHOOSE REPORT` dialog.
2. [Tab] to the width value and change it to "**`80`**".
3. [Tab] to the `RESIZE` action and press [Enter].

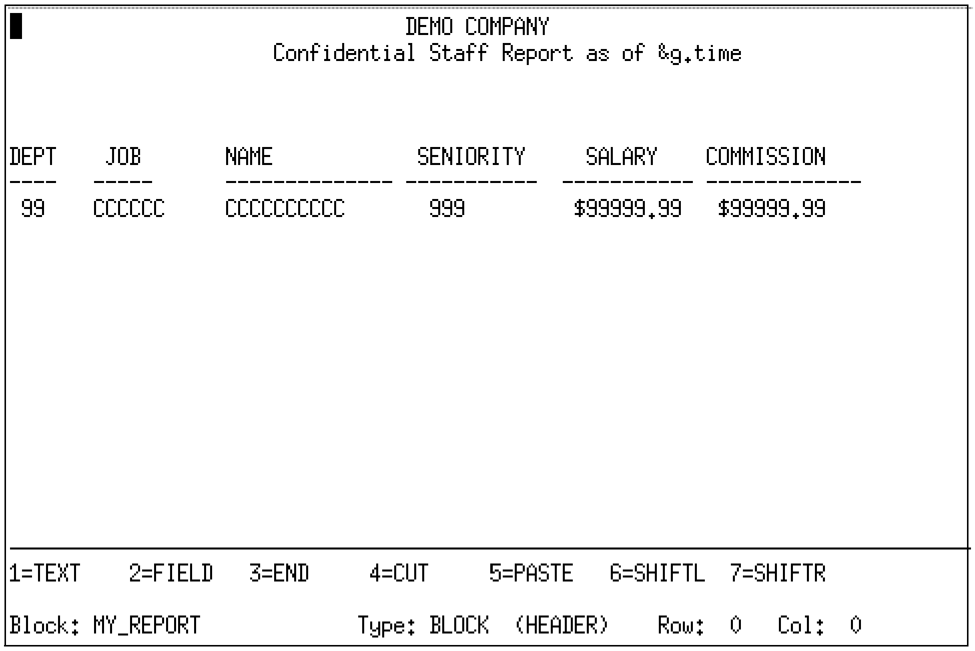
Refine the Report

Refine the presentation with $ signs for the salaries and commissions. Be sure you are in the detail area. (Press [F3] again and then [F4].)

1. Press [F1] to work with text.
2. [Tab] to the SALARY field and add a dollar sign (**`$`**), adjusting the detail column with [F6] or [F7] as appropriate.
3. Move your cursor to the COMMISSION detail and add another **`$`**.
4. Add a report title in the header. ([F3], [F3], [F1], [F1]).
5. Make sure the cursor in the topmost position and press [F4] two or three times to create a few blank lines.
6. Type "**`DEMO COMPANY`**" and press [F6] to center the text.
7. Move the cursor down a line.
8. Type "**`Confidential Staff Report as of &g.time`**" and press [F6] to center.

   `G.TIME` is a global variable that inserts the current date into the report and automatically updates each time you run the report.

Your design should look like this:

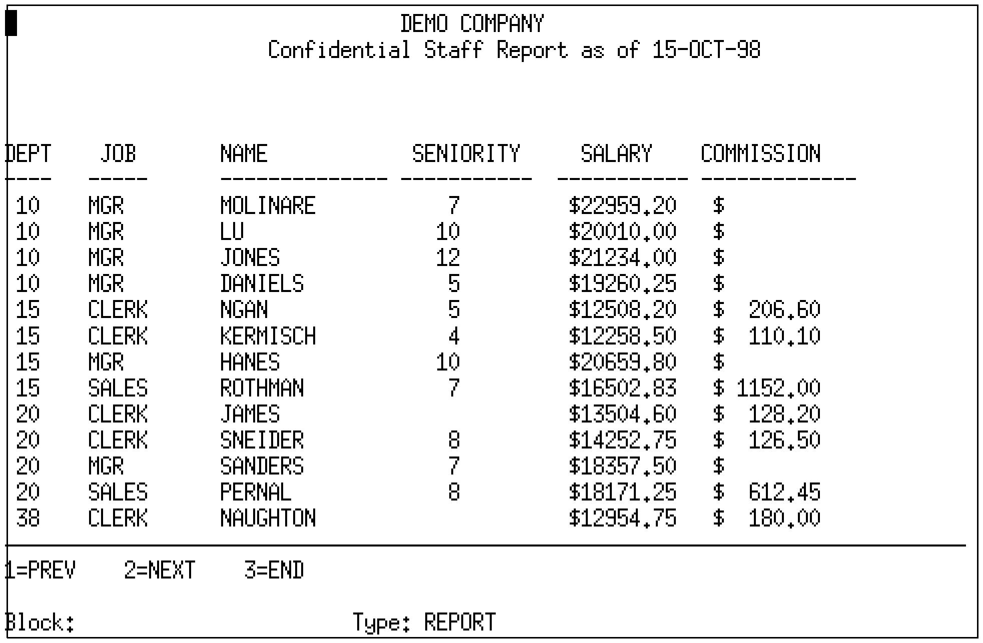


```
                                   DEMO COMPANY
                        Confidential Staff Report as of &g.time



DEPT    JOB       NAME            SENIORITY     SALARY    COMMISSION
----   -----      --------------  -----------  ----------- -------------
 99    CCCCCC     CCCCCCCCCC        999         $99999.99   $99999.99











1=TEXT    2=FIELD   3=END     4=CUT     5=PASTE   6=SHIFTL  7=SHIFTR

Block: MY_REPORT             Type: BLOCK  (HEADER)    Row:  0   Col:  0
```

Run the Report

You can run the report now, if you want to check its appearance.

1. Press [F3] until you get to the CHOOSE REPORT dialog.
2. [Tab] to the RUN action, and press [Enter].



```
                                   DEMO COMPANY
                        Confidential Staff Report as of 15-OCT-98



DEPT    JOB       NAME            SENIORITY     SALARY    COMMISSION
----   -----      --------------  -----------  ----------- -------------
 10    MGR        MOLINARE            7         $22959.20   $
 10    MGR        LU                 10         $20010.00   $
 10    MGR        JONES              12         $21234.00   $
 10    MGR        DANIELS             5         $19260.25   $
 15    CLERK      NGAN                5         $12508.20   $  206.60
 15    CLERK      KERMISCH            4         $12258.50   $  110.10
 15    MGR        HANES              10         $20659.80   $
 15    SALES      ROTHMAN             7         $16502.83   $ 1152.00
 20    CLERK      JAMES                         $13504.60   $  128.20
 20    CLERK      SNEIDER             8         $14252.75   $  126.50
 20    MGR        SANDERS             7         $18357.50   $
 20    SALES      PERNAL              8         $18171.25   $  612.45
 38    CLERK      NAUGHTON                      $12954.75   $  180.00

1=PREV    2=NEXT    3=END

Block:                       Type: REPORT
```

Grouping Information (Creating Breaks and Summations)

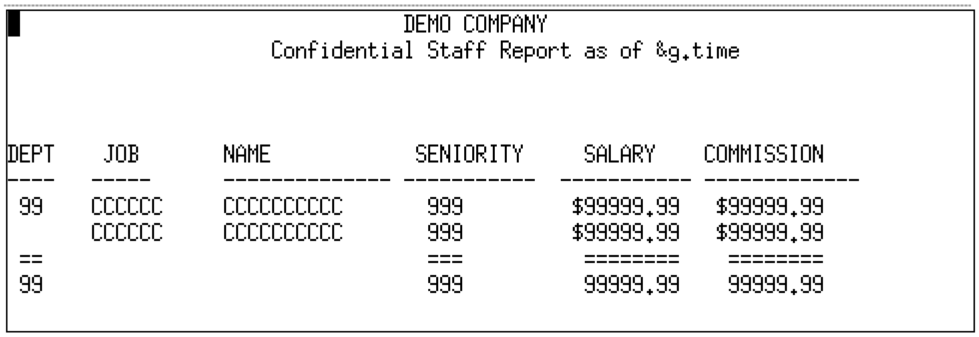
You can create summation fields with SQL or TRIMpl for any numeric column in a report. You can also use TRIMreport's automatic feature, SUMDFLT, to create the summations for entire blocks (CHOOSE REPORTBLOCK > SUMDFLT) or over each break (DEFINE BREAK > SUMDFLT).

SUMDFLT-generated summary fields for breaks typically appear in the footer area. These summations only work for fields that correspond to database detail fields (field structure variables).

```
{
field = SUM(block_name.column_name);
}
```

Now define a report break on the DEPT field. Total the SALARY and COMMISSION for each department without duplicating the department code for each employee.

1. Press [F3], [Tab] to the `MODIFY` action, and press [Enter].
2. In the `CHOOSE REPORTBLOCK` window, [Tab] to the `BREAK` action and press [Enter].
3. Type "**`N`**" in the `Duplicate` field.
4. [Tab] to the `CREATE` action and press [Enter].
5. In the Fields field type "**`DEPT`**"
6. [Tab] to the `SUMDFLT` action and press [Enter] to create a total for each numeric column.



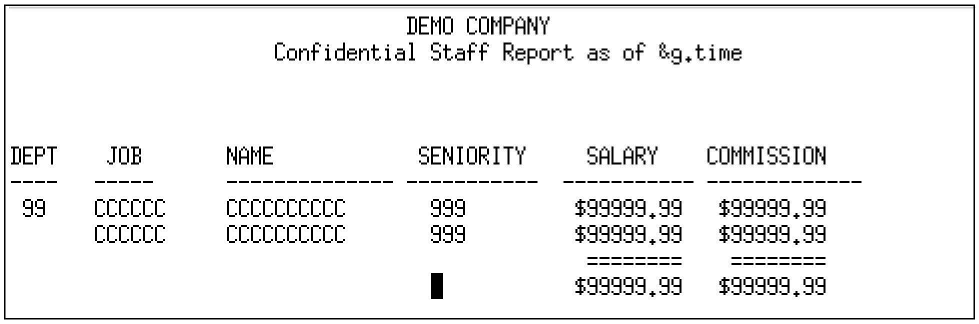
```
                              DEMO COMPANY
                    Confidential Staff Report as of &g.time



DEPT    JOB       NAME           SENIORITY     SALARY    COMMISSION
----   -----    -------------- -----------  ----------- -------------
 99    CCCCCC    CCCCCCCCCC       999        $99999.99   $99999.99
       CCCCCC    CCCCCCCCCC       999        $99999.99   $99999.99
==                                ===         ========    ========
 99                               999         99999.99    99999.99
```

Modify Totals

The default report now breaks for each new DEPT value and the numeric columns — SALARY and COMMISSION — are totalled. The DEPT number is not duplicated for the second NAME, because you specified "N" for duplicates when you defined the break. The DEPT and SENIORITY fields do have totals because they are numeric. Remove the totals. The totals created for breaks actually are in the footer textarea, below the report detail.

1. Press [F3] to return to the `DEFINE BREAK` dialog and [Tab] to `MODIFY` action. Press [Enter].
2. Press [F2].

3. Put the cursor in the footer text under DEPT, 99, and press [F4] to remove the field.
4. Repeat the cut for SENIORITY summation.
5. Press [F1] and use the spacebar to delete the "==" text in the DEPT column and add $ signs in the SALARY and COMMISSION columns.

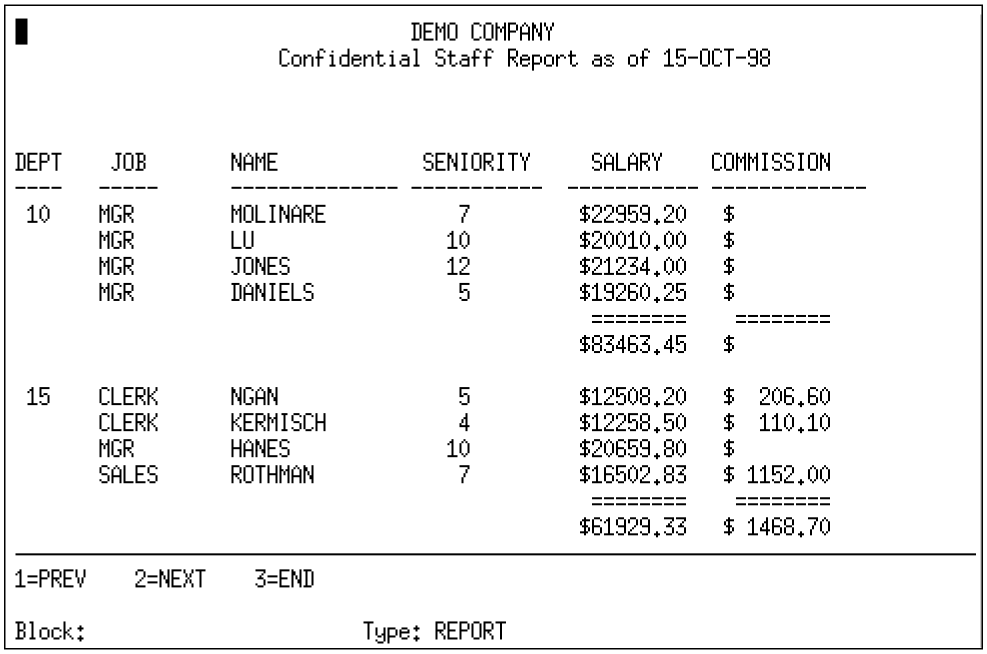


```
                              DEMO COMPANY
                     Confidential Staff Report as of &g.time



DEPT    JOB       NAME           SENIORITY     SALARY    COMMISSION
----   -----      -------------- -----------  ----------- -------------
 99    CCCCCC     CCCCCCCCCC        999        $99999.99   $99999.99
       CCCCCC     CCCCCCCCCC        999        $99999.99   $99999.99
                                                ========    ========
                                    █          $99999.99   $99999.99
```

You now have a professional-looking report that breaks at each department, giving totals of both salary and commission for each department.

Edit the report to provide a total for all departments of DEMO Company.

6. Press [F3] four times to get to the `CHOOSE REPORTBLOCK` dialog for MY_REPORT.
7. [Tab] to the `SUMDFLT` action and press [Enter] to ask for a total for the entire report block.
8. Clean up the report design, adding $s where appropriate and cutting the summary fields for DEPT and SENIORITY.
9. Return to `CHOOSE REPORT` dialog and select the `RUN` action.



```
█                             DEMO COMPANY
                     Confidential Staff Report as of 15-OCT-98



DEPT    JOB       NAME           SENIORITY     SALARY    COMMISSION
----   -----      -------------- -----------  ----------- -------------
 10    MGR        MOLINARE           7         $22959.20   $
       MGR        LU                10         $20010.00   $
       MGR        JONES             12         $21234.00   $
       MGR        DANIELS            5         $19260.25   $
                                                ========    ========
                                               $83463.45   $

 15    CLERK      NGAN               5         $12508.20   $  206.60
       CLERK      KERMISCH           4         $12258.50   $  110.10
       MGR        HANES             10         $20659.80   $
       SALES      ROTHMAN            7         $16502.83   $ 1152.00
                                                ========    ========
                                               $61929.33   $ 1468.70
--------------------------------------------------------------------------
1=PREV    2=NEXT    3=END

Block:                       Type: REPORT
```

You have completed a report with TRIMreport and only written one line of code, "select * from staff order by dept, job".

Printing Your Report

Executing the TRIMreport `RUN` action generates data on your screen. To create a printable report, you must create an executable with TRIMgen and run it with TRIMrun. You can then print the results, *`report_name`*`.out`, to a printer, according to your operating system's procedures. For more information see, "***Running/Printing Reports***" on page 50.

Controlling the Printer

The TRIMpl function `pset()` lets you enter a character string as an X,Y coordinate to change fonts, insert form feeds, and otherwise control the printer's functioning from the reportblocks' control.

The character string is not part of the output buffer and thus occupies no space. It is merely associated with an X,Y (`G.LINENUMBER`, `G.PAGEOFFSET`) position.

When you issue a page break (for example, `paginate(break);`) each character of the output buffer is copied to the output file. Before each character is copied, TRIMreport checks to see if there is an associated string at the current X, Y location. If there is an associated string, it is written to the output file.

`pset()` allows you to send escape codes to a printer. When a report is run within TRIMreport, the function is ignored.

Typical use includes highlighting a field based on a value. For example, the field trigger:

```
{ int highlight;
if (BLK.SALARY 100000) highlight = true;
else                    highlight = false;

if (highlight) pset(-1, -1, "0x1b47");    /* Highlight on     */
field = BLK.SALARY;
if (highlight) pset (-1, -1, "0x1b40");   /* Highlight off    */
}
```

Saving Escape Codes

Since a report may be printed to different output devices, TRIMreport provides a file, `TRIM.CC`, that contains escape codes. Each line in this file ends in a carriage return and contains a symbol (case-insensitive) and an associated escape sequence string. For example,

```
RESET       "0x1c40"
BOLD_ON     "0x1b47"
BOLD_OFF    "0x1b48"
UND_ON      "0x1b2d31"
UND_OFF     "0x1b2d30"
ITAL_ON     "0x1b34"
ITAL_OFF    "0x1b35"
NEC12CPI    "0x1b4d"
NEC10CPI    "0x1b50"
ORANGE      "0x1b7235"
GREEN       "0x1b7236"
BROWN       "0x1b7237"
```

By substituting the symbol for the escape sequence, the field’s trigger code is easier to read:

```
{ int highlight;
if (BLK.SALARY 100000) highlight = true;
else                    highlight = false;

if (highlight) pset(-1, -1, BOLD_ON);    /* Highlight on     */
field = BLK.SALARY;
if (highlight) pset (-1, -1, BOLD_OFF);   /* Highlight off    */
}
```

To change to a new printer with different escape sequences:

- Change the codes in the `TRIM.CC` file.
- Regenerate all the reports with a batch file.

Advanced Procedures

Calling Other Reports or Applications

You can have your report call another report or an application by simply using the `call()` function. This function also has provisions for passing parameters. When one design calls another, the database connections are not severed.

Reports or applications can also return values via the `return()` function in the main trigger.

Making a Report Database-Independent

1. Create a field in the detail area and create a break on that field. Use the following SELECT to retrieve the data from any SQL database:

```
SELECT dept, name, salary
   FROM staff
   ORDER BY dept
```

2. Create a character field, BB, in the detail area with a mask of “A2” and a trigger:

```
{
field = BLK.BB;
}
```

3. Create a variable, BB, in the PRE-BLOCK trigger of the window:

```
{
char BB[2];
;              /* must have a “;” or any expression */
}
```

4. Create a WHERE control that contains the following:

```
set_bb_function()
```

5. Create a SET_BB_FUNCTION that contains the following:

```
{
BLK.BB = substr(BLK.DEPT, 1, 1);
return(1);
}
```

6. Create a BREAK on the BB field.

   The field breaks when the variable changes. The variable is not an internal node variable and must be created in the PRE-BLOCK trigger.

7. Change the value of BB before the break check occurs by using the WHERE control. The WHERE control is invoked after each row is returned but before anything else happens, including the break check.

The code in the SET_BB_FUNCTION causes a break to occur when the first digit of *dept* changes, but it can be any algorithm that generates a new value when a break is needed.

The assignment of a value to the field is inconsequential. Also, you must create a variable in the PRE-BLOCK trigger because the break references it.

Specifying a Single Pagetext Header/Footer

You can have all blocks in a report use the same pagetext header and footer by letting all blocks inherit the pagetext header and footer areas from the parent block.

For a block to be generated and, therefore, callable from a trigger, it must be one of the following:

- First block in a design.
- Sibling of a generated block.
- Child of a generated block that contains a SELECT statement.

For the pagetext footer and header to be automatically inherited by all blocks, the parent block must contain pagetext header and footer text and include a SELECT statement (with CREATE, DEF_HORZ, DEF_VERT, or DEF_VER2) or a list reference.

In the SELECT statement:

```
SELECT 1 FROM table
```

In a list reference

```
[G.DATA_LIST]
```

If you don't include a SELECT statement, the main report trigger generates an error message when it calls the child block. For example,

```
INTERNAL: exec BLK
```

The block was never generated (because the parent block did not have a SELECT). Adding a SELECT and creating the block (CREATE, DEF_HORZ, DEF_VERT, or DEF_VER2 action), although inconvenient, solves the problem and does not affect performance since the parent block is never called —it exists solely to provide a standard pagetext header and footer.

NOTE: *The main trigger must be changed to call the first child block rather than its parent to avoid executing the parent block's SELECT statement.*

Fields to Center Headings

Fields are useful to generate a centered heading for a report. For example, you may want an indentifying title to appear on each of the several similar reports. Create a field in the pagetext header area with a trigger like this:

```
{
field = getenv(DIVISION_NAME_SYMBOL);
}
```

When the report is run and the field is executed, the content of `DIVISON_NAME_SYMBOL` is copied to the output buffer. If the field is centered, the data is centered in the pagetext header area. Centering the field in the pagetext header area centers everything within the field automatically.

Transforming Values

You can suppress data in a field c or translated it to another value. For example, a block uses the following SELECT for certain *job* values (for example, "MGR") for which the values in the *salary* column are suppressed:

```
SELECT name, job, salary
   FROM staff
```

The detail area for the block by default contains

```
 NAME                JOB                SALARY
CCCCCC            CCCCCC                $99,999

{                 {                 {
field=BLK.NAME;   field=BLK.JOB;     field=BLK.SALARY;
}                 {                  {
```

You suppress the specified value in the job column — in the following example, "MGR" — by changing the *salary* field's trigger to

```
{
if (BLK.JOB != "MGR") field = BLK.SALARY;
}
```

Multiple Columns from One Field

To change a field's X and Y positions when writing to the output buffer you use the special variables `G.LINENUMBER` and `G.PAGEOFFSET`.

```
{ int i;
for (i=1; i<5; i++) {
   field         = i;
   G.PAGEOFFSET   = G.PAGEOFFSET + 10;
```

```
    G.LINENUMBER++;
    }
}
```

The trigger example above causes a *single* field to generate the following four lines of output:

```
1
     2
          3
               4
```

This is extremely useful for reports that allow dynamic numbers of columns. By manipulating `G.LINENUMBER` and `G.PAGEOFFSET`, *field* can write to any location within the page dimensions.

NOTE: *G.LINENUMBER and G.PAGEOFFSET do not change fields that belong to the pagetext header or footer areas.*

Second Tutorial: Two-Block Report

You have now followed the procedure to create, design, and format a very basic report with only one report block. Typically reports require more than a single block to collect and communicate the information that people need. This tutorial provides an example of a two-block report and demonstrates how to use different features of TRIMreport.

You should be comfortable with TRIMreport navigation to complete this tutorial, since it shows more advanced features of TRIMreport and takes advantage of shortcuts.

The report shows personnel in each department and their sales and commission totals.

Create Report with One Block

Create a report using two tables ORG and STAFF , which have a one to many relationship.

1. `CHOOSE REPORT` dialog, complete the Name, Creator, page length, and width fields. > `CREATE`.
2. `CHOOSE REPORTBLOCK` dialog, complete Name: `ORG`.
3. [Tab] to `CREATE` [Tab]> `SELECT`.
4. `DEFINE SELECT`> [F4], type “`select * from org`”> `DEF_HORZ`.
5. [F3] > [F3] to design window.
6. Clean up the report beginning with Detail: [F4], [Tab], [F2].
7. DEPTNUMB only requires 2 digits, MANAGER 3 digits.

   [Tab] to `NEXT` and press [Enter] to change the next field.
8. After the 5th Field, [F3] to return to design window and move the columns. ([F6]).
9. Adjust the header text over each column ([F1], [F1]). Don’t forget to scroll to the right for “leftover” text or you may get an error message if you try to resize your report.

Create Second Block

Now create a child block that shows information from the STAFF table. This child block inherits attributes, including pagetext from its parent.

1. CHOOSE REPORTBLOCK > CHILD > Type "`STAFF`"> `CREATE`.

   Notice that that the status line at the bottom of the window says "Block: STAFF" and you see the "parent" design in the design window.

2. [F5] > `SELECT` > [F4] > Type "`select * from staff where dept = &p.deptnumb`" > `DEF_HORZ`.

   The & indicates a variable replacement for the value. The current deptnumb value from ORG is substituted for each select (one per detail line) before it goes to the database. You could type "parent.deptnumb" or "ORG.deptnumb".

3. [F3] > [F3] to see the second block detail in your report. Clean up the appearance the same way you tidied ORG columns and headers.

   Dept only needs 2 digits, ID and Year 3 digits. In this exercise, put the $ for SALARY and COMM inside the DEFINE FIELD Mask field so that the symbol travels with the values. When you've completed all 7 fields, press [F3].

4. Indent the STAFF detail under Dept. Name to give the report a visual hierarchy. ([F7])
5. Clean up the header text ([F1], [F1]).
6. Fiddle with the header and detail lines until they are pleasing. Add two blank lines in the STAFF header text.
7. [F3] > [F3] > [F3] > [F3] > `FILE` > "filename."
8. `RUN`.
9. To further modify STAFF, from the `CHOOSE REPORTBLOCK`> `LIST` and choose `STAFF` from the list that appears.

Create Sum Values for Second Block

1. Select STAFF block. In `CHOOSE REPORTBLOCK` for Staff, select `SUMDFLT`.
2. Remove the summations that don't make sense, leaving only the Salary and Commission.
3. Create an explanatory footer: "Total for dept:" and create a field to display the department number.
4. `[F2]` > `[F1]` > Type "`Total for dept:`" > `[F3]` > `[F2]`.
5. Type "`DEPT`" > `NEWFIELD`.
6. `DEFINE FIELD` as number, accept right justify, mask of two digits (99).
7. `TRIGGER` > [F4] > Type "` field = p.deptnumb;`" (TRIMpl code)
8. [F3] > [F3] > [F3] > [F3] > [F3] > `FILE` > `RUN`.
9. Don't forget to resize the report.

☞ Add Report Title and Pagenumbering

1. CHOOSE `REPORTBLOCK` > `PAGETEXT` > `MODIFY` > `[F1]` > `[F1]`
2. Insert 3 lines [F4] x3, then type "`Department Report as of`" press [F6]
3. Type "&g.time" press [F6].
4. [F3] > [F3] > [F2] > [F1] > [F3] > [F1].
5. Type "Page: &g.pagenumber" > `[F6]`
6. [F3] > [F3] > [F3] > [F3] > [F3] > `FILE` > `RUN`.

☞ Add Continuation Information

Notice that when you're looking at the report, if you press [F2] you see that the report is more than a single page long.

1. [F3] > `MODIFY` > `LIST` > `STAFF`.
2. [F3] > `PAGETEXT` > `MODIFY` > `YES` > [F1] > [F1].
3. Replace existing text with "continued for department &p.deptnumb". [F6]
4. [F3] > [F3] > [F3] > [F3] > [F3] > `FILE` > `RUN`.

☞ Changing G.TIME Format

With all the recent attention being paid to Year 200 issues, you may want to format your report date more specifically, just to keep everyone comfortable in your office.

1. [F3] > `MODIFY` (ORG) > `PAGETEXT` > `MODIFY` > [F1] > [F1].
2. Delete the text "&.gtime".
3. [F3] > [F2] > Type "DATE" in the Name: field > NEWFIELD.
4. Complete the dialog box as follows: Data type is DATE, accept default justification, Mask is HH:MI:SS DD-MON-YYYY.
5. TRIGGER > [F4] > Type "field = SYSDATE;"
6. [F3] > [F3] > [F6] to adjust field location.
7. [F3] > [F3] > [F3] > [F3] > `FILE` > `RUN`.

   Pagetext now that it has been changed is no longer inherited. If you wanted the pagetext for STAFF to also have a date, you would have to follow the same procedure for the staff pagetext header.

After completing the step-by-step in "***Designing a Report***" on page 32 and this tutorial, you have executed nearly all the actions and commands of TRIMreport.

Chapter 5
Running Your Report

Running/Printing Reports

To run a report you use TRIMrun with the report name. You can use options to specify report performance and output. TRIMrun executes compiled code, which makes the applications, reports, and stand-alone applications interactive. It is also called the "runtime."

```
trimrun [runfile] [DB-login] [out-file] [options] [-p parms]
```

The runfile must have a `.run` file name extension.

You must specify DB-login only if the user is going to interact with a database. You can also specify the database connection in the application.

You use `out_file` option with report designs to direct output to the file name given. This example command creates a report file named `myrep.out`:

```
trimrun myreport myname/mysecret myrep
```

Options

-d*n*	Debug error number *n*. Use -d with no error number to see all errors.
-f *string*	Specifies a string to use for the form feed. The default is a universal form feed command. To specify "no form feed" use the switch with no string argument.
-k *filename*	Input (playback) keys from filename.
-m	Displays memory usage (number of bytes allocated.)
-o *filename*	Output (record) keys to file *filename.*
-t *filename*	Shows execution trace where *filename* is output file; if no *filename* is specified, trace goes to stdout.
-p	Allows information to be passed to the application on the command line. You must put the switch at the end of the command followed by all the parameters (everything following this option is assumed to be a parameter). This example passes the string myname into the myapp application: `trimrun myapp -p myname`

Output Buffer

When a report is run, output is written to a buffer that represents the page size originally defined in the `CHOOSE REPORT` window (generally 66 lines by 80 characters). The buffer, a virtual page, is copied to the output file whenever a `paginate()` function call is made with the break option. For example:

```
paginate(break);
```

You can write to the virtual page until a page break is performed. A field can move up (`G.LINENUMBER--`) and down the page (`G.LINENUMBER++`).

When a "*field* =" is performed, the value is written to the current X, Y (`G.LINENUMBER, G.PAGEOFFSET`) position, overwriting whatever is there.

The `pset()` function call does not overwrite, but inserts the text at the position at which it is called.

When a pagebreak occurs (via `paginate(break);`) the output buffer is written to the output file.

The copying process starts from the top of the output buffer (`G.LINENUMBER = 0, G.PAGEOFFSET =0`); then proceeds to the right and down, after checking to see if `pset()` placed a string at the current position. If `pset()` placed a string, that string is copied to the output buffer. Then, it copies the character at the current position and moves to the next character.

Output Files

The output buffer is written to the output file whenever a page break is performed. The buffer is written to the last output file opened with an `open()` or `append()` function call; when a second `open()` or `append()` is called, it closes the first. If a file is passed to either function, the file is opened and the output buffer is written to it. For example,

```
{
open ("tst.out");
...
```

If an `open()` function is called in an empty string, the report writes to the standard output device (terminal screen). For example:

```
{
char file-name[10];
file-name = " ";
open (file-name);
...
```

The report can contain only an `open()` function call with no parameters:

```
{
open();
...
```

Then, if a name is passed to a report on the command line, all output is written to the file *outfile.OUT:*

```
trimrun report-name userid/password outfile.OUT
```

If no name is passed to the report on the command line, the report is opened and the output written to a file using the same name as the report with an `.OUT` extension. For the previous example, if *outfile.OUT* is not given, all output is written to the file *report.name.OUT*. If the output file specified is null, then the data is written to the screen.

When running a report from within TRIMreport, the `open()and pset()` function calls are ignored.

Two other functions, `log()` and `list_file()` can write to a file. Read the *TRIMpl Function Reference* for details on each function.

Parameters

When a report is run, parameters can be passed to it either from a command line or from another report or application design using `call()`.

Generally, the parameters serve to qualify the SELECT statements in report blocks. For example, a date can be passed to a report that prints only events occurring on that date.

Passing a large number of parameters can be clumsy when using a construct, such as *variable=parm[32];* because of the exact ordering it necessitates. Also, referencing a parm[*n*] that has not been passed generates an error.

An easier way to pass a large number of parameters is by using a *list* variable. In the following example a list variable is created with two columns;

ID_KEY	VALUE
RPTDAT	07-04-66
CLASS	88
SECURITY	1
REPORT_NAME	REUNION.REPORT.LIS

The list is loaded with rows that contain an ID_KEY and a VALUE.

When the report is called, the list of variables is passed in. The report uses the `extract()` function to retrieve the values from the list. The following is an example of the main trigger of such a report:

```
{
date rptdat;               /* declare variables              */
int class;                 /* to receive values              */
int security;              /* from passed in list            */
char report_name[32];
if (count(parm)) {         /* if true, list is passed in     */
   rptdat      = extract(parm[0], "RPTDAT");
   class       = extract(parm[0], "CLASS");
   security    = extract(parm[0], "SECURITY");
   report_name = extract(parm[0], "REPORT_NAME");
   }
else {                     /* no parms, so test case         */
   rptdat      = "01-MAY-97";
   class       = 0;
   security    = 1;
   report_name = "tst.out";
   }
…
```

An ID string (RPTDAT, CLASS, and so on) is passed to the `extract()` function that uses it to find the correct row in the list. It then returns data from the VALUE column of the list row. The `extract()` function can be designed to generate an error or return a default value.

Also, notice that there is a check to see if any parameters have been passed:

```
if (count(parm)) …
```

This switch allows the report to test values if it is run from within TRIMreport or stand-alone for debugging.

Chapter 6
TRIMreport Screens

The following diagram shows the basic functional interaction between TRIMreport screens.

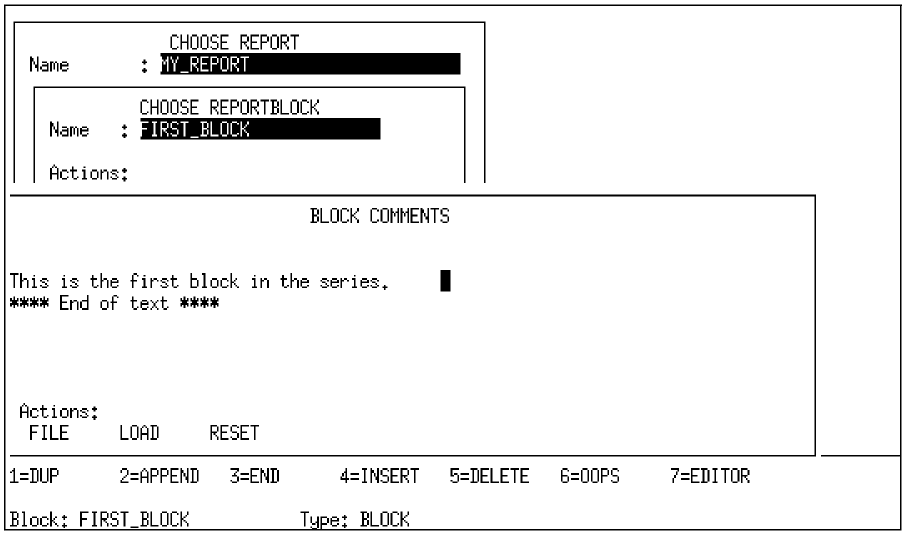
The screens are presented in alphabetical order, each screen beginning a new page.

BLOCK COMMENTS

Navigation: CHOOSE REPORT > CHOOSE REPORTBLOCK > COMMENT

Use the work area to enter comments about the current report block or the main report node. Use **[PgUp/PgDn]** and arrow keys to scroll text.

The report and reportblock must already be created.



```
               CHOOSE REPORT
 Name        : MY_REPORT
             CHOOSE REPORTBLOCK
   Name   : FIRST_BLOCK

   Actions:
                              BLOCK COMMENTS

This is the first block in the series.
**** End of text ****



 Actions:
  FILE     LOAD     RESET
1=DUP      2=APPEND   3=END      4=INSERT   5=DELETE   6=OOPS     7=EDITOR

Block: FIRST_BLOCK          Type: BLOCK
```

Actions

FILE	Store comments in an operating system file.
LOAD	Load comments from an operating system file. Text lines are inserted before the line at the top of the screen.
RESET	Erase all text lines.

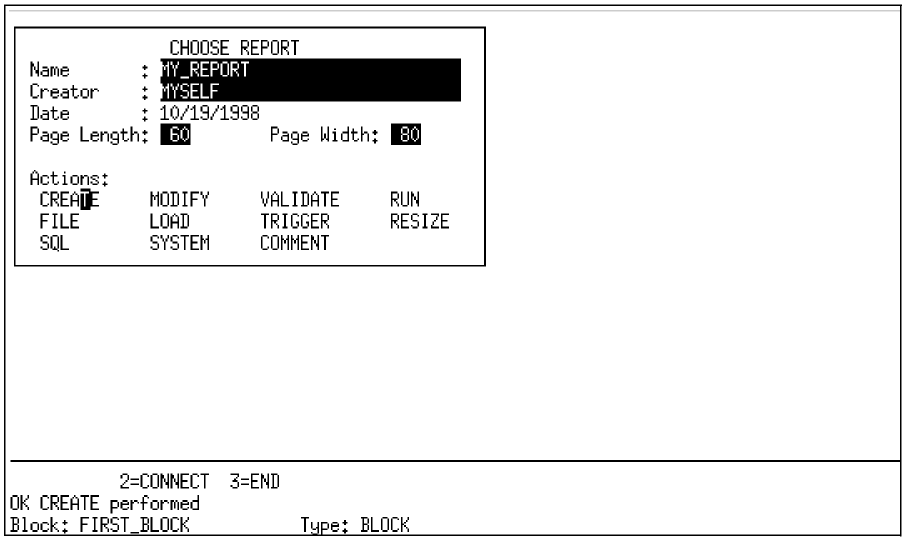
Function Keys

F1 DUP	Duplicate the line under the cursor.
F2 APPEND	Append a new line below the cursor.
F3 END	Return to the previous screen.
F4 INSERT	Insert a new line above the cursor.
F5 DELETE	Delete the line under the cursor.
F6 OOPS	Insert the previously deleted line above the cursor.
F7 EDITOR	Escape to the operating system editor defined by EDITOR environment variable.

CHOOSE REPORT

Navigation: AT START

Use this dialog (displayed on the starting screen) to enter general information for the report and execute global actions.



```
               CHOOSE REPORT
Name       : MY_REPORT
Creator    : MYSELF
Date       : 10/19/1998
Page Length: 60        Page Width: 80

Actions:
 CREATE     MODIFY     VALIDATE     RUN
 FILE       LOAD       TRIGGER      RESIZE
 SQL        SYSTEM     COMMENT

           2=CONNECT  3=END
OK CREATE performed
Block: FIRST_BLOCK          Type: BLOCK
```

Fields

Name	Report name. This can be any combination of alphanumeric characters. TRIMreport uses the name as the default file name if the report is filed.
Creator	Identifier of the person creating the report.
Date	Automatically filled with current date.
Fixed Header Rows	Number of text rows in header. The number of rows specified are for the printed report.
Fixed Footer Rows	Number of text rows in footer. The number of rows specified are for the printed report.

Actions

CREATE	Create the main tree node with the dialog information and displays the CHOOSE REPORTBLOCK dialog.
MODIFY	Display the CHOOSE REPORTBLOCK dialog.
VALIDATE	Validate the report definition. When an error is detected in a trigger during validation, the DEFINE TRIGGER screen is displayed with the cursor positioned at the error.
RUN	Execute the current report design
FILE	File or discard the current report tree.
LOAD	Load previously saved report files.

TRIGGER	Create a report trigger; display the DEFINE TRIGGER screen.
PROFILE	Modify report profile items such as default editor, default number and data formats.
SQL	Define and execute SQL commands except SELECT statements.
SYSTEM	Define and execute system commands.
COMMENT	Enter comment text for the report design.
RESIZE	Enter new values for page length and width.

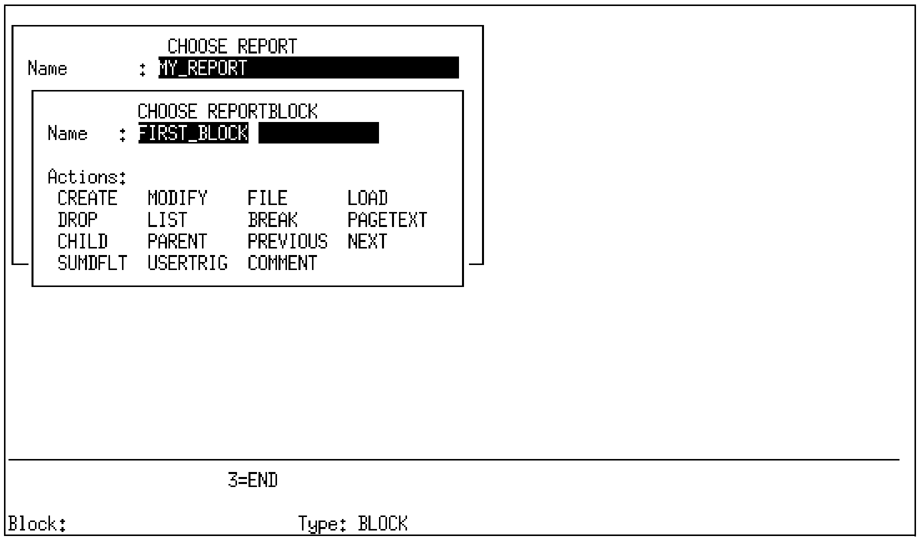
Function Keys

F2 CONNECT	Connect to the database.
F3 END	Exit from DVreport.

CHOOSE REPORTBLOCK

Navigation: `CHOOSE REPORT > CREATE | MODIFY`

Use this dialog to manage the blocks that make up a report. Each report block must have a unique name entered in the Name field.



```
               CHOOSE REPORT
 Name       : MY_REPORT
            CHOOSE REPORTBLOCK
   Name   : FIRST_BLOCK

   Actions:
    CREATE   MODIFY   FILE     LOAD
    DROP     LIST     BREAK    PAGETEXT
    CHILD    PARENT   PREVIOUS NEXT
    SUMDFLT  USERTRIG COMMENT

                     3=END

Block:                          Type: BLOCK
```

Field

Name	Name of the report block to create or modify.

Actions

CREATE	Create the report block using the given name. Display the screen painter main screen.
MODIFY	Modify the report block identified by Name. Display the screen painter main screen.
FILE	File the report block including its child and sibling report blocks.
LOAD	Load a previously filed report block at this point in the design. The name of the report block must not conflict with an existing name. DVreport attempts to resize the subtree to match the current page size.
DROP	Delete the report block and the underlying subtree, if any.
LIST	Display a list of all available report blocks.
BREAK	Display the DEFINE BREAK dialog.
PAGETEXT	Display the DEFINE PAGETEXT dialog.
CHILD	Replace current report block with its child report block.
PARENT	Replace current report block with its parent report block.
PREVIOUS	Replace the current report block with the previous (left sibling) report block.

NEXT	Replace the current report block with the next (right sibling) report block.
SUMDFLT	Create the default summation fields in the report block footer.
SUMDFLT	Display the CHOOSE TRIGGER dialog.
COMMENT	Display the BLOCK COMMENTS dialog.

Function Key

F3 END	Return to the CHOOSE REPORT dialog.

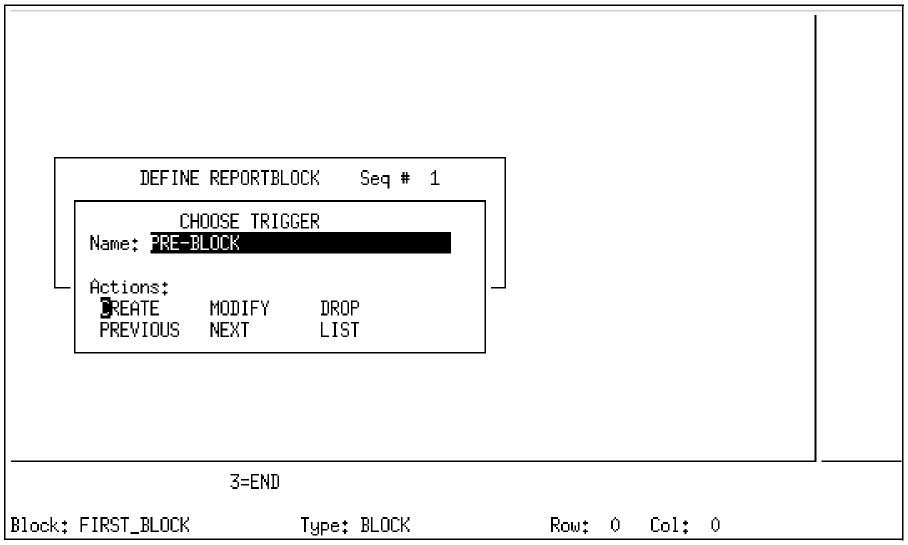
CHOOSE TRIGGER

Navigation: `CHOOSE REPORT > CHOOSE REPORTBLOCK > CREATE | MODIFY > TRIGGER`

Use this screen to create a new trigger, or modify or delete an existing trigger. Triggers are automatically defined for each:

- Report (CHOOSE REPORT)
- Report block (DEFINE REPORT BLOCK)
- Page text (DEFINE PAGETEXT)
- Report block break (DEFINE BREAK)
- Field (DEFINE FIELD)

You can define additional triggers (CHOOSE REPORTBLOCK) that are accessible to all other triggers in the report.



```
      DEFINE REPORTBLOCK    Seq #  1
           CHOOSE TRIGGER
  Name: PRE-BLOCK

  Actions:
   CREATE     MODIFY     DROP
   PREVIOUS   NEXT       LIST

                  3=END
Block: FIRST_BLOCK          Type: BLOCK             Row:  0   Col:  0
```

Predefined Triggers

The following table shows the triggers and report elements for which they are valid.

Valid For	Trigger	Description
Report Blocks and Report Block Breaks	PRE-BLOCK	Execute once before anything else in the report block.
	POST-BLOCK	Execute once after all else in the report block.
	POST-HEADER	Execute once after the report block header is written.
	PRE-FOOTER	Execute once before the footer is written.
Report Blocks	PRE-FETCH	Execute before data is retrieved.
	POST-FETCH	Execute after each data retrieval.

Valid For	Trigger	Description
	CHILD	Execute once before the PRE-FOOTER trigger.
Fields	FIELD	Execute once each time the field is accessed in sequence.
Page text	PAGINATE	Execute once each time a page break is required.
The Report	MAIN	Execute once before anything else is done in the report.

Field

NAME	Name of the trigger.

Actions

CREATE	Create the user-defined trigger identified in the Name field. Display the DEFINE TRIGGER screen.
MODIFY	Display the DEFINE TRIGGER screen.
DROP	Delete the current trigger identified in the Name field.
PREVIOUS	Replace the current trigger with the previous trigger.
NEXT	Replace the current trigger with the next trigger.
LIST	List the currently defined triggers.

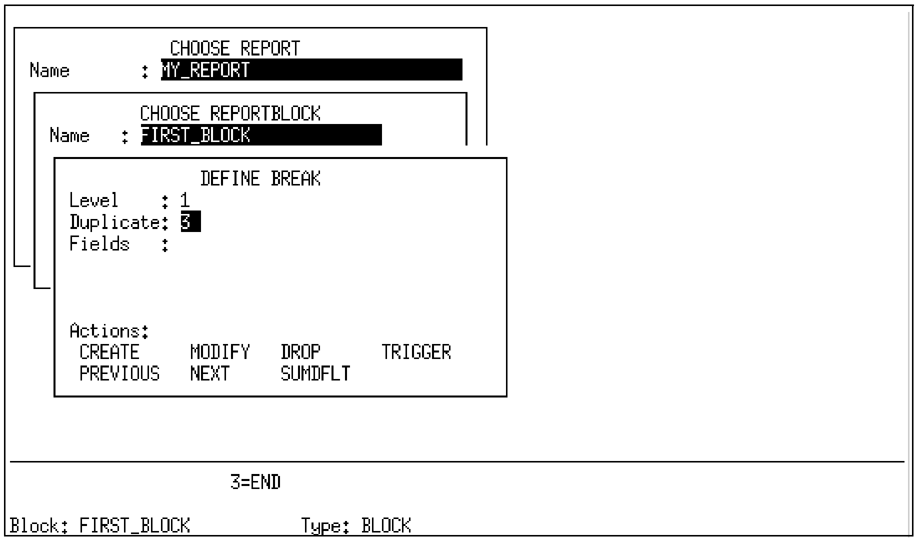
Function Key

F3 END	Return to the previous screen.

DEFINE BREAK

Navigation: `CHOOSE REPORT > CHOOSE REPORTBLOCK > BREAK`

Use this dialog to define one or more break levels for the current report block. You must choose Yes or No in the Duplicate field execute a CREATE action before using DEFINE BREAK.



```
            CHOOSE REPORT
Name      : MY_REPORT

          CHOOSE REPORTBLOCK
 Name   : FIRST_BLOCK

                DEFINE BREAK
  Level    : 1
  Duplicate: S
  Fields   :


  Actions:
   CREATE     MODIFY   DROP      TRIGGER
   PREVIOUS   NEXT     SUMDFLT

                        3=END
Block: FIRST_BLOCK           Type: BLOCK
```

Fields

Name	Name of the break being defined.
Duplicate	Y to duplicate break fields; N to not duplicate break fields.
Fields	List of break fields.

Actions

CREATE	Create a report block break; open the Name and Fields fields for entry.
MODIFY	Display the screen painter main screen.
DROP	Delete the report block break.
TRIGGER	Display the CHOOSE TRIGGER dialog.
PREVIOUS	Replace the current report block break with its parent report block break.
NEXT	Replace the current report block break with its child report block break.
SUMDFLT	Create the default summation fields in the report block break footer.

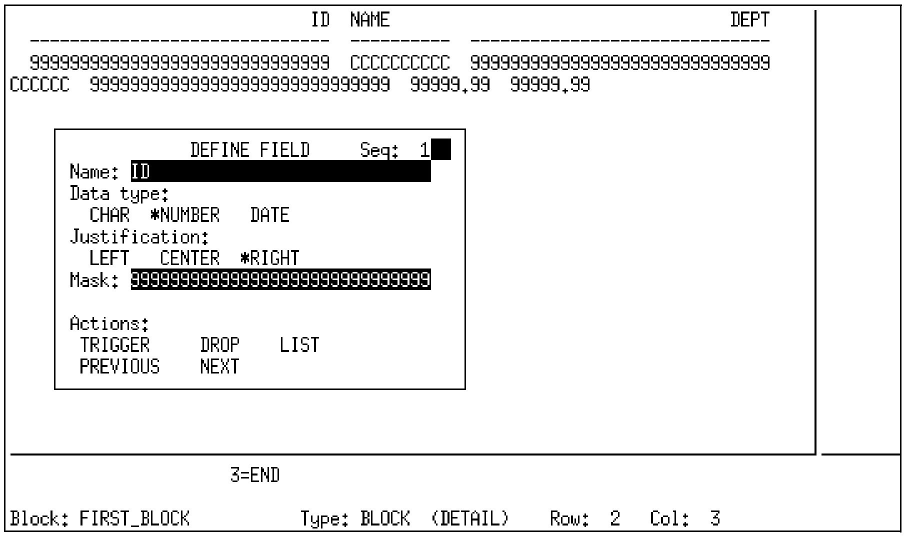
Function Key

F3 END	Return to the CHOOSE REPORTBLOCK dialog

DEFINE FIELD

Navigation: `CHOOSE REPORTBLOCK > MODIFY > F4 > F2`

Use this screen to define a field in the header, detail, or footer area of a report block, or the header or footer of a report block break or page text.



```
                              ID  NAME                                DEPT
  ------------------------------  ----------  ------------------------------
  999999999999999999999999999999  CCCCCCCCCC  999999999999999999999999999999
CCCCCC  999999999999999999999999999999  99999.99  99999.99

                DEFINE FIELD     Seq:  1
      Name: ID
      Data type:
        CHAR  *NUMBER   DATE
      Justification:
        LEFT   CENTER  *RIGHT
      Mask: 999999999999999999999999999999

      Actions:
       TRIGGER     DROP    LIST
       PREVIOUS    NEXT

                      3=END

Block: FIRST_BLOCK          Type: BLOCK  (DETAIL)    Row:  2   Col:  3
```

Fields

Seq	Sequence number of the field
Name	Name of the field; any combination of up to 30 characters.
Mask	Formatting mask for the field. (Consult the appendixes for more information.)

Actions

TRIGGER	Display the DEFINE TRIGGER dialog.
DROP	Delete field from the current report block.
LIST	List the currently existing fields; field in list can be selected for definition. LIST is not available for a new field.
PREVIOUS	Replace current field with previous field.
NEXT	Replace current field with previous field.

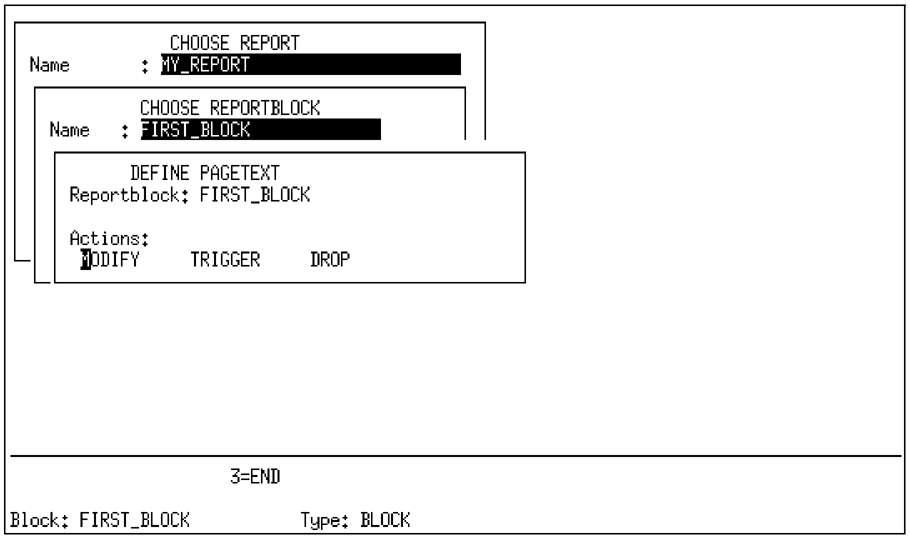
Function Key

F3 END	Return to the previous screen.

DEFINE PAGETEXT

Navigation: `CHOOSE REPORT > CHOOSE REPORTBLOCK > PAGETEXT`

Use this dialog to define the page header, footer, and trigger for the current report block and its child report blocks. PAGETEXT is inherited from the parent report block. If the parent PAGETEXT is modified, each child PAGETEXT automatically receives the modifications except when the child PAGETEXT was individually defined.



```
              CHOOSE REPORT
Name       : MY_REPORT
           CHOOSE REPORTBLOCK
  Name   : FIRST_BLOCK
        DEFINE PAGETEXT
  Reportblock: FIRST_BLOCK

  Actions:
   MODIFY     TRIGGER     DROP

                    3=END

Block: FIRST_BLOCK          Type: BLOCK
```

Field

Name	Name of the PAGETEXT

Actions

MODIFY	Return to the previous screen.
TRIGGER	Edit the page header and footer text. Display the screen painter main screen.
DROP	Delete the page header, footer, and trigger entries.

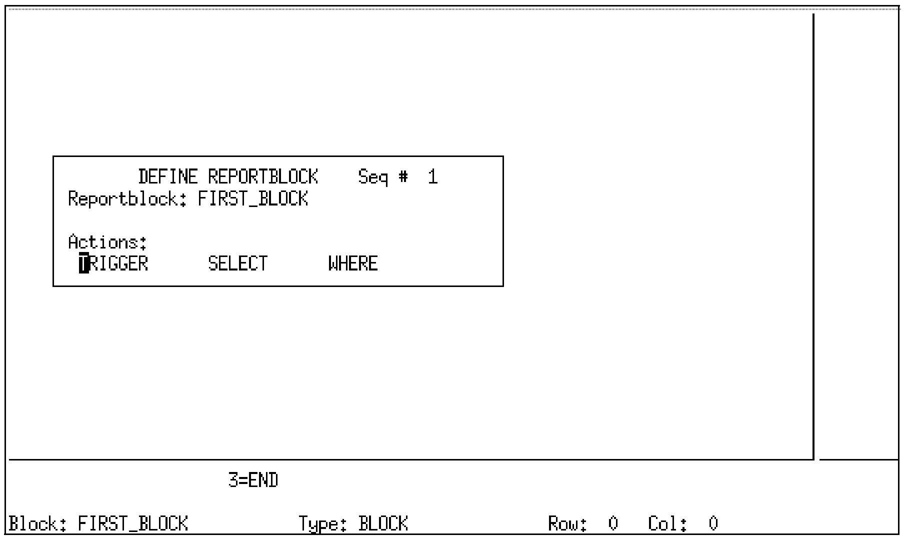
Function Key

F3 END	Return to the CHOOSE REPORTBLOCK dialog.

DEFINE REPORTBLOCK

Navigation: `CHOOSE REPORT > CHOOSE REPORTBLOCK > CREATE|MODIFY > F5`

Use this dialog to edit the code for the report block trigger, the SELECT command, or the WHERE clause.



```
      DEFINE REPORTBLOCK    Seq #  1
Reportblock: FIRST_BLOCK

Actions:
 TRIGGER      SELECT      WHERE

                    3=END
Block: FIRST_BLOCK          Type: BLOCK              Row:  0   Col:  0
```

Field

Reportblock	Name of the report block.

Actions

TRIGGER	Display the CHOOSE TRIGGER screen.
SELECT	Display the DEFINE SELECT screen.
WHERE	Display the DEFINE WHERE screen.

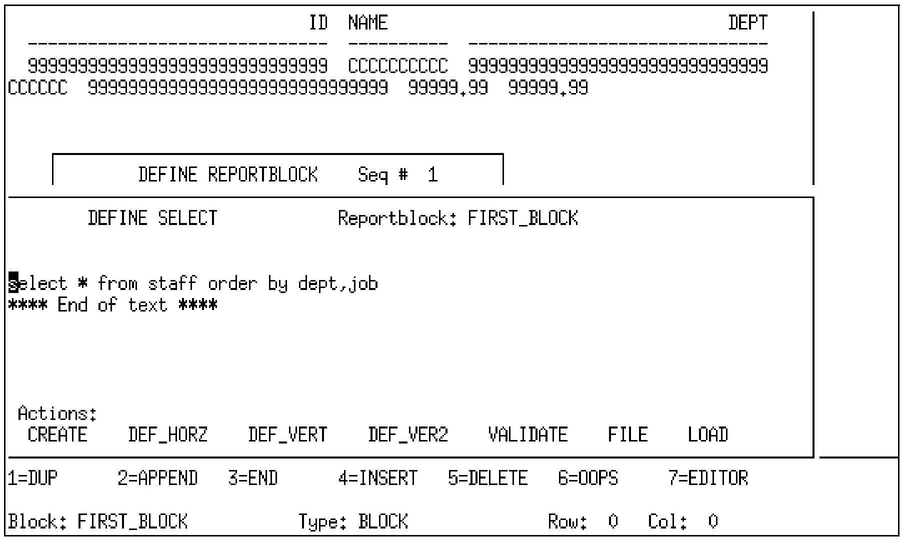
Function Key

F3 END	Return to the CHOOSE REPORTBLOCK dialog.

DEFINE SELECT

Navigation: `CHOOSE REPORTBLOCK > CREATE | MODIFY > F5 > SELECT`

Use this screen to edit the SELECT statement for the report. You can scroll text using the [PgUp/PgDn] and arrow keys.



```
                              ID  NAME                                 DEPT
  ------------------------------  ----------  ------------------------------
  999999999999999999999999999999  CCCCCCCCCC  999999999999999999999999999999
CCCCCC  999999999999999999999999999999  99999.99  99999.99


    DEFINE REPORTBLOCK    Seq #  1

       DEFINE SELECT           Reportblock: FIRST_BLOCK


select * from staff order by dept,job
**** End of text ****



 Actions:
  CREATE    DEF_HORZ    DEF_VERT    DEF_VER2    VALIDATE    FILE    LOAD
1=DUP      2=APPEND   3=END      4=INSERT   5=DELETE   6=OOPS     7=EDITOR

Block: FIRST_BLOCK          Type: BLOCK            Row:  0   Col:  0
```

Field

Reportblock	Name of the current report block.

Actions

CREATE	Perform SQL DESCRIBE with the SELECT statement; no fields are created.
DEF_HORZ	Perform SQL DESCRIBE with the SELECT statement; default fields for a horizontal report are created.
DEF_VERT	Perform SQL DESCRIBE with the SELECT statement; default fields for a vertical report are created.
DEF_VER2	Perform SQL DESCRIBE with the SELECT statement; default fields are created and aligned for a vertical report.
VALIDATE	Validate the SQL statement syntax and binding.
FILE	Store entire SELECT statement in an operating system file.
LOAD	Load SELECT statement from an operating system file. The statement is inserted before the line at the top of the screen.

Function Keys

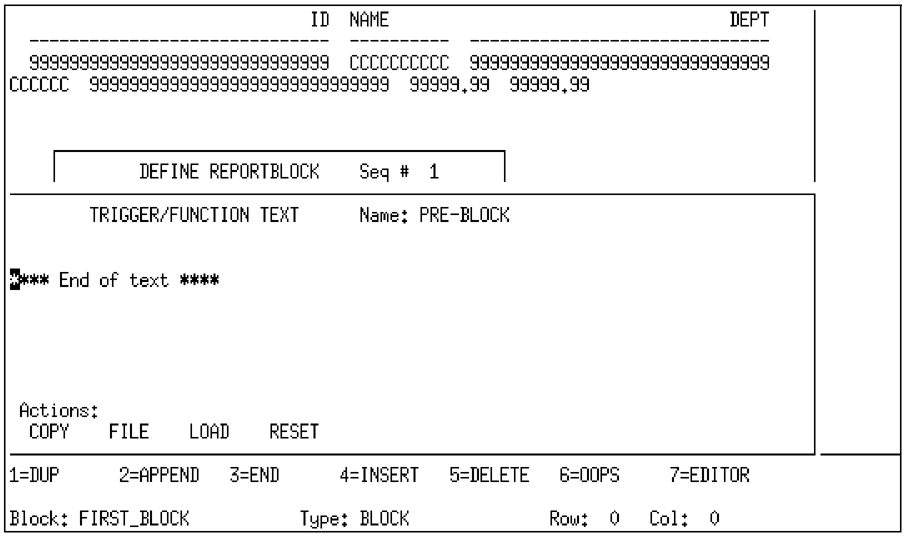
F1 DUP	Duplicate the line under the cursor.
F2 APPEND	Append a new line below the cursor.

F3 END	Return to the previous screen.
F4 INSERT	Insert a new line above the cursor.
F5 DELETE	Delete the line under the cursor.
F6 OOPS	Insert the previously deleted line above the cursor.
F7 EDITOR	Escape to the operating system editor definied by the EDITOR environment variable.

DEFINE TRIGGER

Navigation: `CHOOSE REPORTBLOCK > CREATE | MODIFY > F5 > TRIGGER > CREATE | MODIFY`

Use this screen to edit the code for all triggers. Use the [PgUp/PgDn] and arrow keys to scroll text.


```
                              ID  NAME                               DEPT
  ------------------------------  ----------  ------------------------------
  999999999999999999999999999999  CCCCCCCCCC  999999999999999999999999999999
CCCCCC  999999999999999999999999999999  99999.99  99999.99


        DEFINE REPORTBLOCK    Seq #  1

       TRIGGER/FUNCTION TEXT      Name: PRE-BLOCK


**** End of text ****




 Actions:
  COPY    FILE    LOAD    RESET
1=DUP      2=APPEND   3=END      4=INSERT   5=DELETE   6=OOPS     7=EDITOR

Block: FIRST_BLOCK          Type: BLOCK             Row:  0   Col:  0
```

Field

Name	Name of the trigger to define.

Actions

COPY	Copy trigger code from an archive library. Code is inserted before the line at the top of the screen.
FILE	Store entire trigger code in an operating system file.
LOAD	Load trigger code from an operating system file. Code is inserted before the line at the top of the screen.
RESET	Erase all trigger code.

Function Keys

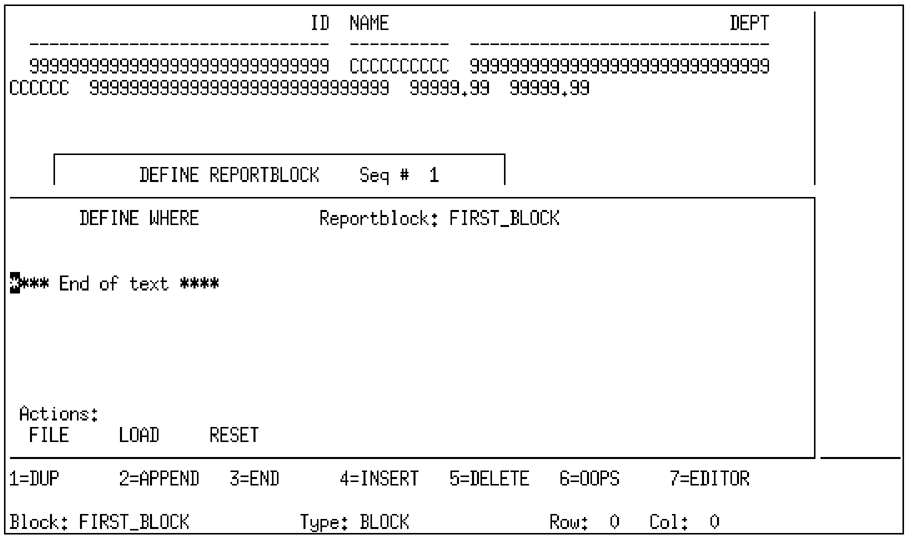
F1 DUP	Duplicate the line under the cursor.
F2 APPEND	Append a new line below the cursor.
F3 END	Return to the previous screen.
F4 INSERT	Insert a new line above the cursor.
F5 DELETE	Delete the line under the cursor.
F6 OOPS	Insert the previously deleted line above the cursor.
F7 EDITOR	Escape to the operating system editor definied by the EDITOR environment variable.

DEFINE WHERE

Navigation: `CHOOSE REPORTBLOCK > CREATE | MODIFY > F5 > WHERE`

Use this screen to edit the optional WHERE clause for the current report block SELECT statement. If the condition in the WHERE clause is true for a row of data, the current row is included; if false, the row is not included and the next row is evaluated.

Scroll text using the [PgUp/PgDn] and arrow keys.



```
                              ID  NAME                              DEPT
  ------------------------------  ----------  ------------------------------
  999999999999999999999999999999  CCCCCCCCCC  999999999999999999999999999999
CCCCCC  999999999999999999999999999999  99999.99  99999.99


          DEFINE REPORTBLOCK    Seq #  1

       DEFINE WHERE           Reportblock: FIRST_BLOCK

**** End of text ****



 Actions:
  FILE     LOAD     RESET

1=DUP      2=APPEND   3=END      4=INSERT    5=DELETE    6=OOPS     7=EDITOR

Block: FIRST_BLOCK          Type: BLOCK            Row:  0   Col:  0
```

Field

Reportblock	Name of the current report block.

Actions

FILE	Store the WHERE clause in an operating system file.
LOAD	Load the WHERE clause from an operating system file. The WHERE clause is inserted before the line at the top of the screen.
RESET	Delete all lines.

Function Keys

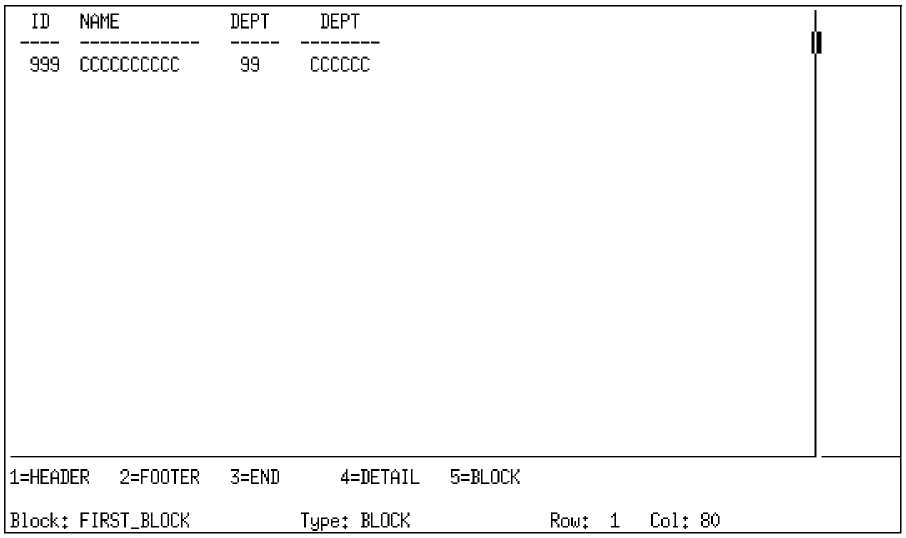
F1 DUP	Duplicate the line under the cursor.
F2 APPEND	Append a new line below the cursor.
F3 END	Return to the previous screen.
F4 INSERT	Insert a new line above the cursor.
F5 DELETE	Delete the line under the cursor.
F6 OOPS	Insert the previously deleted line above the cursor.
F7 EDITOR	Escape to the operating system editor definied by the EDITOR environment variable.

Screen Painter

The screen painter defines the physical layout of the report. You can edit header, detail, and footer areas as well as manipulate the fields in all areas. Only the areas associated with the current report block can be edited. The current report block name is displayed in the status line; the row and column of the cursor's position are also displayed.

Main Screen

Use this screen to display the current report design. Select the area to edit by pressing the appropriate function key. The main screen can be scrolled using the [PgUp/PgDn], [F9] (Right), [F10] (Left), and arrow keys. The function keys available at this point vary depending on the dialog from which the screen painter was accessed.



```
  ID   NAME          DEPT     DEPT
 ----  ------------  -----  --------
  999  CCCCCCCCCC     99     CCCCCC

1=HEADER   2=FOOTER   3=END      4=DETAIL   5=BLOCK

Block: FIRST_BLOCK          Type: BLOCK            Row:  1   Col: 80
```

Function Keys (from MODIFY PAGETEXT or DEFINE BREAK)

F1 HEADER	Edit the header text area.
F2 FOOTER	Edit the footer text area.
F3 END	Return to previous dialog.

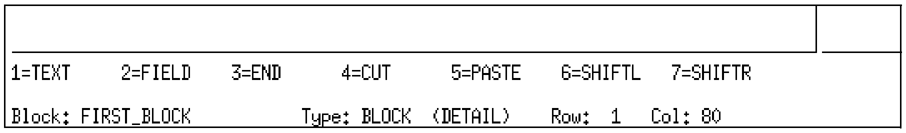
Function Keys (from CHOOSE REPORTBLOCK)

F1 HEADER	Edit the header text area.
F2 FOOTER	Edit the footer text area.
F3 END	Return to the CHOOSE REPORTBLOCK dialog.
F4 DETAIL	Edit the detail text area.
F5 BLOCK	Display the DEFINE REPORT BLOCK dialog.

Edit Screen

Use this screen to manipulate the fields in the chosen area (header, footer, detail). Use Tab and arrow keys to move the cursor between fields.

From the main screen select header, footer, or detail. The working area's appearance appears the same but the actions at the bottom changes to:



```
1=TEXT     2=FIELD    3=END      4=CUT      5=PASTE    6=SHIFTL   7=SHIFTR

Block: FIRST_BLOCK          Type: BLOCK  (DETAIL)    Row:  1   Col: 80
```

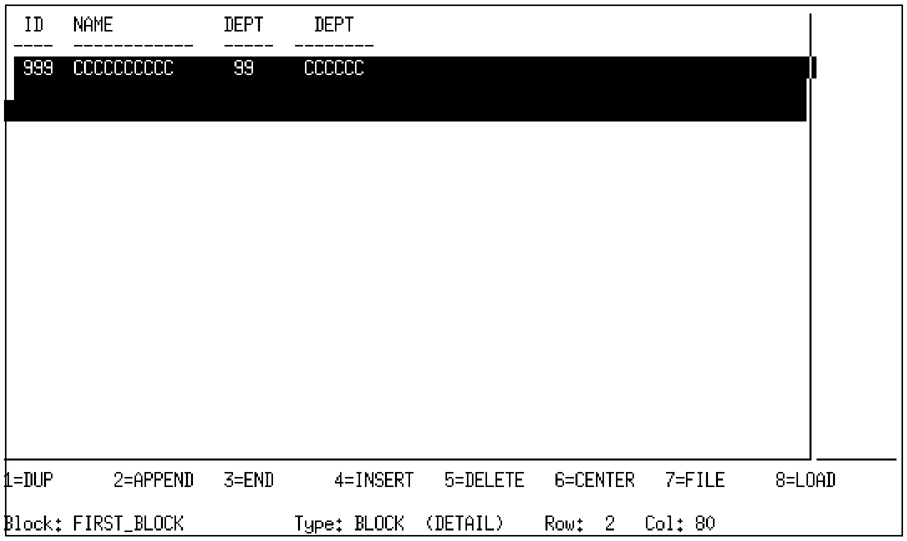
Function Keys

F1 TEXT	Display the screen painter text edit screen.
F2 FIELD	If the cursor is within a defined field, display the DEFINE FIELD dialog. If the cursor is not within a defined field, display the CHOOSE FIELD dialog for a new field.
F3 END	Return to the screen painter main screen.
F4 CUT	Cut the field in which the cursor is located.
F5 PASTE	Retrieve a field that was cut or marked and places it at the current cursor location. Multiple fields may be cut and pasted in last-in-first-out order. For example, cut *field-A*, cut *field-B*, cut f*ield-C*, paste *field-C*, paste *field-B*, paste *field-A*.
F6SHIFTL	All fields to the left of the cursor are moved one space left each time the function key is pressed.
F7SHIFTR	All fields to the right of the cursor are moved one space right each time the function key is pressed.

Edit Text Screen

Use this screen to modify the text of the previously chosen area: header, footer, detail. Use Tab and arrow keys to move the cursor between fields.

From the edit screen, choose F1, text and the editable area becomes highlighted and the actions change:



```
 ID   NAME          DEPT     DEPT
----  ------------  -----  --------
 999  CCCCCCCCCC     99     CCCCCC


1=DUP     2=APPEND   3=END     4=INSERT   5=DELETE   6=CENTER   7=FILE     8=LOAD

Block: FIRST_BLOCK          Type: BLOCK  (DETAIL)    Row:  2   Col: 80
```

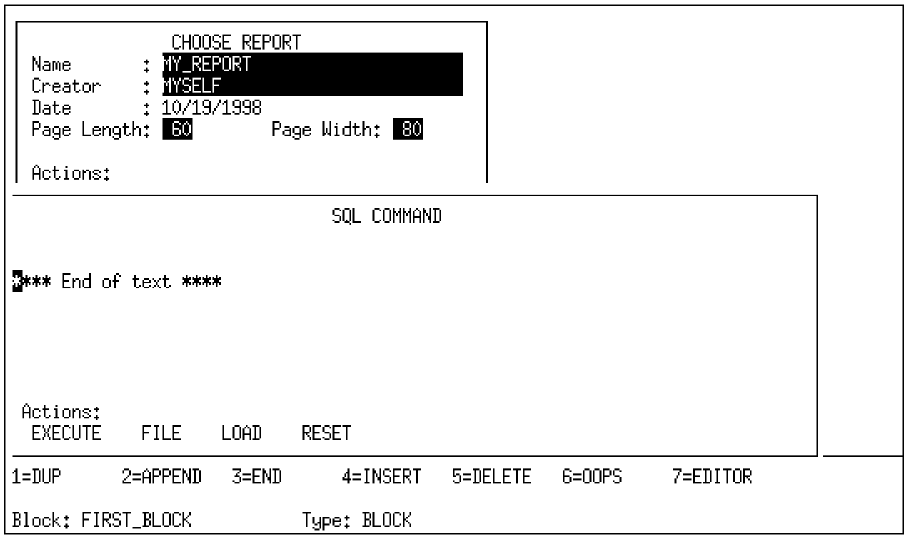
Function Keys

F1 DUP	Duplicate the current line.
F2 APPEND	Append a new line below the cursor.
F3 END	Return to the previous screen.
F4 INSERT	Insert a new line above the cursor.
F5 DELETE	Delete the current line.
F6 CENTER	Center the line in which the cursor is located.
F7 FILE	Save the entire text to an operating system file.
F8 LOAD	Load text from an operating system file; the text is placed above the cursor.

SQL COMMAND

Navigation: `CHOOSE REPORT > SQL`

Use the displayed work area for entering SQL commands. Scroll text using the [PgUp/PgDn] and arrow keys.



```
                 CHOOSE REPORT
 Name       : MY_REPORT
 Creator    : MYSELF
 Date       : 10/19/1998
 Page Length: 60        Page Width: 80

 Actions:

                                SQL COMMAND

**** End of text ****

 Actions:
  EXECUTE    FILE    LOAD    RESET

1=DUP      2=APPEND   3=END      4=INSERT   5=DELETE   6=OOPS     7=EDITOR

Block: FIRST_BLOCK          Type: BLOCK
```

Actions

EXECUTE	Execute the SQL command(s).
FILE	Store SQL command(s) in an operating system file.
LOAD	Load SQL command(s) from an operating system file. Commands are inserted before the line at the top of the screen.
RESET	Erase all commands.

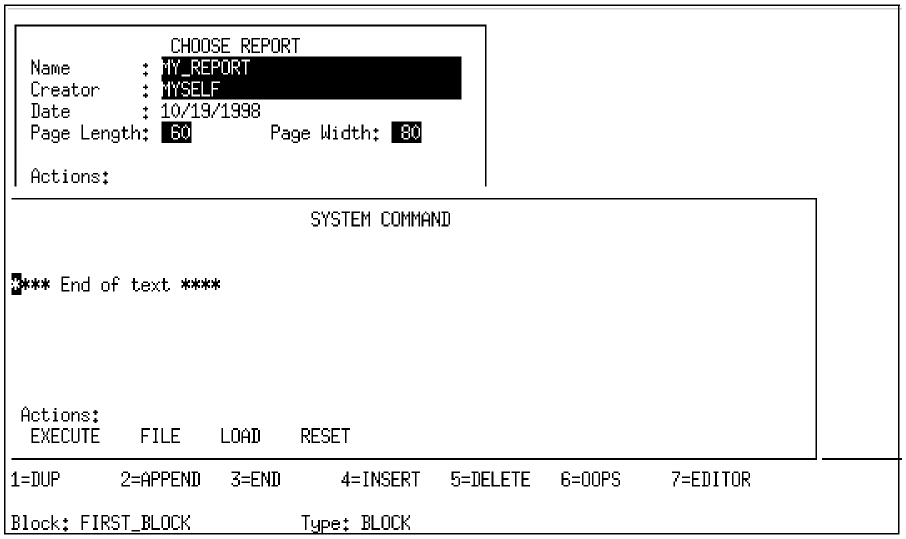
Function Keys

F1 DUP	Duplicate the line under the cursor.
F2 APPEND	Append a new line below the cursor.
F3 END	Return to the previous screen.
F4 INSERT	Insert a new line above the cursor.
F5 DELETE	Delete the line under the cursor.
F6 OOPS	Insert the previously deleted line above the cursor.
F7 EDITOR	Escape to the operating system editor definied by the EDITOR environment variable.

SYSTEM COMMAND

Navigation: `CHOOSE REPORT > SYSTEM`

Use this screen to execute system commands from within DVreport . You can enter multiple commands, one command per line. Scroll text with [PgUp/PgDn] and arrow keys.



```
                CHOOSE REPORT
 Name       : MY_REPORT
 Creator    : MYSELF
 Date       : 10/19/1998
 Page Length: 60        Page Width: 80

 Actions:
                               SYSTEM COMMAND

**** End of text ****

 Actions:
  EXECUTE    FILE    LOAD    RESET
1=DUP      2=APPEND   3=END      4=INSERT   5=DELETE   6=OOPS     7=EDITOR

Block: FIRST_BLOCK          Type: BLOCK
```

Actions

EXECUTE	Execute the command(s).
FILE	Store command(s) in an operating system file.
LOAD	Load system command(s) from an operating system file. Commands are inserted before the line at the top of the screen.
RESET	Erase all commands.

Function Keys

F1 DUP	Duplicate the line under the cursor.
F2 APPEND	Append a new line below the cursor.
F3 END	Return to the previous screen.
F4 INSERT	Insert a new line above the cursor.
F5 DELETE	Delete the line under the cursor.
F6 OOPS	Insert the previously deleted line above the cursor.
F7 EDITOR	Escape to the operating system editor definied by the EDITOR environment variable.

Appendix A
Format Masks

Format masks let you specify how you want string data to appear. These format mask are based loosely on Oracle's format masks.

char

The *char* datatype has only one format mask: `A`*n* where *n* specifies the width.

numeric

Numeric data has a wide selection of formatting options. You can specify the width of a field or resulting conversions by the length of the mask.

Character	Description
%	Percent sign at right of number.
$	Dollar sign at left of number.
B	Display zero as blank.
0	Display leading zeros.
9	A digit position.
other	Delimiting character (not leading)

Formatting Examples

Value	Mask String	Result
1958	9,999.99	1,958.00
1958	099999	001958
56.789	$999.99	$56.79
0	999.99	0.00
0	099.99	000.00
0	B99.99	
4083692300	999/999-9999	408/369-2300

datetime

Datetime data has a variety of formatting options. You specify the width of the field or resulting conversions by the length of the mask.

Sequence	Description	Use (I/O)
YYYY	Four-digit year	I/O
YY	Two-digit year	I/O
RR	Two-digit year in another century.	O
MM	Two-digit month of year (01 – 12)	I/O
MON	Three-character month (all upper case)	I/O
mon	Three-character month (all lower case)	I/O
Mon	Three-character month (first letter upper case)	I/O
MONTH	Fully named month (all upper case)	I/O
month	Fully named month (all lower case)	I/O
Month	Fully named month (mixed case)	I/O
DDD	Three digit day of year (001 – 366)	I/O
DD	Two digit day of month (01 – 31)	I/O
D	Single-digit day of week (1 – 7)	O
DY	Three character day (all uppercase)	I/O
dy	Three character day (all lower case)	I/O
Dy	Three-character day (1st letter upper case)	I/O
DAY	Fully-named day (all upper case)	I/O
day	Fully-named day (all lower case)	I/O

Sequence	Description	Use (I/O)
Day	Fully-named day (mixed case)	I/O
HH12	Two-digit hour (00 – 11)	I/O
HH,HH24	Two-digit hour (00 – 23)	I/O
MI	Two-digit minutes (00 – 59)	I/O
SS	Two-digit seconds (00 – 59)	I/O
SSSS	Seconds past midnight (0000 – 86399)	I/O
J	Julian day	I/O
Q	Single-digit quarter of year (0 – 4)	O
W	Single-digit week of month (1 – 4)	O
WW	Two-digit week of year (01 – 52)	I/O
other	Delimiting character	I/O

Appending *th* (case insensitive) to any uppercase digit mask appends *ST*, *ND*, *RD*, or *TH* depending on the last digit. For lowercase digit masks the lowercase version is appended.

Put embedding characters that are valid masks inside double quotes (").

The default mask is **DD-MON-YY**.

Formatting Examples

Value	Mask String	Result
Feb 6, 1958	DD/MM/YYYY	06/02/1958
Feb 6, 1958	qth quarter of YY	1st quarter of 58
Nov 1, 1995 20:48:46	HH12:MI on Day	08:48 on Wednesday

User-defined

You can use a special format mask when built-in format masks don't meet your need. For example, when the database represents YES/NO as 1 and 0, none of the built-in masks adequately translate the values.

A user-defined format mask looks like

```
Unnfun [ ( parm[2] [, …, parm[n] ] ) ]
```

where

nn	represents the width of the field in characters. The screen painter represents this mask with the character “u,” as in UUUU.

func	is the user-defined trigger responsible for the input and output from and to the field. func() has two implicit parameters: • parm[0] false on output, true on input • parm[1] output: the actual field; input: the string entered.
parm[2, 3, …, *n*]	represents the optional parameters you can pass with the user-defined trigger.

The user-defined trigger is called every time the system reads or rights to the _D variable associated with the field variable and must return the following;

- ***output*** — any data.
- ***input (query)*** — any data.
- ***input***— data with the same datatype as field.

Examples

A Y[es]/N[o] field where 1 is yes and 0 is no. Based on language used, the corresponding letters should be used, but the actual values are either 0 or 1.

```
Format mask:
   ‘U1yes_no’.

   Function/Trigger yes_no:
   if (parm[0])        /* input ?     */
      return(decode(G.lan,“SWE”,decode(parm[1],”J”,1,”N”0,-1),
                             “ENG”,decode(parm[1],”Y”,1,”N”,0,-1));

   else                /* output      *
      return(decode(G.lan,”SWE”,decode(parm[1]0,”N”,”J”),
                             “ENG”,decode(parm[1]0,”N”,”Y”),”?”));
```

Symbols
& 48
in textareas 11
&p.deptnumb 48
/ 11

A
actions
CREATE 33
DEF_HORZ 34
DEF_VERT 19
DEF_VERT2 19
MODIFY 36
ampersand
in textareas 11
applications
calling from report 44

B
block detail
report block 9
SELECT control 9
WHERE control 9
break footer
SUM fields 15
break levels
defining 62
multiple 14
break sub-blocks 14
break trigger 13
breaks
creating 41
SUMDFLT 15

C
call() 44
case-sensitivity
Sybase databases 34
changing
format masks 37
child blocks
creating 48
CHOOSE REPORT 6, 13, 35
CHOOSE REPORT dialog 32
CHOOSE REPORTBLOCK 8
SUMDFLT 41
user triggers 13
columns
removing 37
commenting
current report block 55
main report note 55
control sections
defined 7
controls
SELECT 9
WHERE 10
count() 13
CREATE 9, 33
creating
break levels 62
breaks 41
child blocks 48
groups 41
new fields 48
report blocks 58
summation 48
summations 41
totals 41
triggers 60

D
database
conserving resources 18
DATE 49
date
variable 39
DEF_HORZ 9, 19, 34
DEF_VER2 9
DEF_VERT 9, 19
DEF_VERT2 19
defaults 4
summations 41
DEFINE BREAK 13, 14
SUMDFLT 41
DEFINE FIELD 13, 37, 48
DEFINE PAGETEXT 13
DEFINE REPORTBLOCK 13
defining
break levels 62
report fields 63
report text 64
deleting
triggers 60
design methods 17
list-driven 17
select-driven 17
trigger-driven 17
designing
reports (see layout) 70
detail area
column names 19
defined 7
dialogs
CHOOSE REPORT 32, 35
DEFINE FIELD 37
displaying
function results 11
time and date 11
distribution
workload 17

E
editing
report areas 71
report text 72
SELECT statements 66
trigger code 68
embedded variables
see variables
embedding functions 11
entering SQL commands 73
error
truncate text 47
errors
function calls 13
internal mismatch 10
non-SELECT statements 9
passing parameters 13
escape codes
for printing 43
executing
global actions 56
system commands 74
execution sequence
report blocks 7
execution summary
report blocks 14

F
field structures 9
field trigger 13
fields 12
creating 48
footer
defined 7
defining report 63
pagetext 8
footer defining report 63
form feed
suppressing or changing 7
format masks
changing 37
moving 37
formatting
pages 6
forward slash 11
functions
call() 44
embedding 11
identifying 11
paginate() 7
pset() 43
return() 44
using 11

G
G. LINENUMBER 8
G.LINENUMBER 43, 46
G.PAGEOFFSET 43, 46
G.TIME 11, 39
global actions
executing 56
groups
creating 41

H
headers
defined 7
defining report 63
pagetext 8
removing 37
how to
call other reports/apps 44
change format masks 37
create a report 32

modify reports 36
move format masks 37
save a report format 36
save escape codes for printing 43
two-block report 47
use variables 48

I

inheritance
pagetext 7
report blocks 64
INSERT, DELETE 9

L

layout
reports 70
leftover text 47
LIST 49
list-driven design 17

M

main report note
commenting 55
main trigger 13
managing
report blocks 58
triggers 60
masks
datetime 49
MODIFY 36
modifying 41
page text 49
report 36
report block triggers 65
report blocks 58, 64
totals 41
triggers 60
moving
format masks 37

N

naming
report blocks 58
NEWFIELD 48
numeric fields
breaks 15

O

Oracle
tutorial directions 32
using 38
organization
report blocks 5
output buffer 6

P

page
formatting 6
PAGE LENGTH
max values 6
page size
resizing 39
page width
max values 6
with Oracle 32
pagetext 8, 13, 49
inheriting 7
PAGETEXT action
see also pagetext
pagetext settings
modifying 8
paginate( ) 7
parameters
errors 13
passing 13
parent-child 7
report blocks 5
POST-BLOCK trigger 15
execution summary 14
POST-FETCH trigger
execution summary 14
POST-HEADER trigger 15
execution summary 14
PRE-BLOCK trigger 15
execution summary 14
PRE-FOOTER
execution summary 14
trigger 15
printer
controlling 43
printing
escape codes 43
reports 43
pset() 43

R

removing
columns 37
headers 37
report
creating 32
report block 13
defined 7
organization 5
report blocks
block detail 9
commenting 55
execution sequence 7
execution summary 14
inheritance 64
managing 58
organization 7
parent-child 7
siblings 7
report design
requirements 17
report text
editing 72
reports
calling others 44
editing areas 71
layout 70
printing 43
screen painter 70
RESIZE
action 39
resizing 47
page size 39
return() 44
running
reports 43

S

saving
report format 36
screen painter 70
SELECT
editing 66
SELECT control 9
SELECT statement
execution summary 14
SELECTcontrol
block detail 9
select-driven design 17
semicolons 9, 10
siblings 7
report blocks 5
specifying
multiple break levels 14
SQL
entering commands 73
sub-blocks
break 14
SUM fields
break footer 15
SUMDFLT 41, 48
CHOOSE REPORTBLOCK 41
DEFINE BREAK 41
in breaks 15
summations
creating 41
using 48
Sybase
case sensitivity 34
SYSDATE 49
system commands
executing 74

T

terminating SQL 9
time
variable 39
totals 41
creating 41
trigger controls
subblock 15
trigger-driven design 17
triggers
break 13
editing code 68
field 12, 13
main 13
managing 60
modifying report block 65
page text 13
pagetext 8

POST-BLOCK 15
PRE-BLOCK 15
predefined 60
PRE-FOOTER 15
report block 13
trim.cc 43
truncate text
error message 47

U
UPDATE 9
user triggers 13

V
variables 11
assigning in WHERE control 10
builtin 11
G.LINENUMBER 43
G.PAGEOFFSET 43
G.TIME 39
identifying 11
replacement 48
WHERE control 10

W
WHERE control 10
assigning variables 10
block detail 9
declaring variables 10
execution summary 14
using semicolons 10
window
see also dialogs