MULTICS REPORT WRITER
REFERENCE MANUAL

SUBJECT

Description of the Multics Report Writer

SOFTWARE SUPPORTED

Multics Software Release 11.0

ORDER NUMBER

GB63-00

January 1985

Honeywell

PREFACE

This manual describes the Multics Report Writer which was developed from the LINUS report writer. The Multics Report Writer provides the capabilities to utilize formatted data extracted from sources other than Multics Relational Data Store data bases. The only user-visible change is the default page length which was changed from 66 to 0, resulting in unpaginated reports by default.

This manual presupposes some basic knowledge of the Multics system, and does not attempt to provide information covered in either of the following two manuals: NEW USERS' INTRODUCTION TO MULTICS - PART I and PART II, Order No. CH24 and CH25 respectively.

This manual contains references to the MULTICS PROGRAMMER'S REFERENCE MANUAL (Order No. AG91) -- referred to in this text as "Programmer's Reference Manual", the MULTICS SUBROUTINES and I/O MODULES (Order No. AG93) -- referred to as "Subroutines Manual", and the MULTICS LOGICAL INQUIRY and UPDATE SYSTEM REFERENCE MANUAL (Order No. AZ49) -- referred to as "LINUS Manual".

The manual is divided into several sections which are outlined below.

Sections 1 and 2
: contains overview and tutorial information which should be read by anyone intending to use the Report Writer directly, to produce formatted reports.

Section 3
: contains overview and tutorial information on the subroutine interface to the Report Writer. This section should be read by anyone who intends to write a Multics subsystem that includes the Report Writer.

Section 4
: contains descriptions of Report Writer requests. This section should be read by anyone intending to use the Report Writer.

Section 5
: contains descriptions of Report Writer subroutine entrypoints.

Section 6
: contains descriptions of Table Manager subroutine entrypoints.

Section 7
: contains a PL/I example that uses the subroutine interface to the Report Writer. Sections 5 and 6 should be read by anyone who intends to write subsystems that include the Report Writer.

The information and specifications in this document are subject to change without notice. This document contains information about Honeywell products or services that may not be available outside the United States. Consult your Honeywell Marketing Representative.

©Honeywell Information Systems Inc., 1985
File No.: 1L13 GB63-00

CONTENTS

Section 1 Report Writer System 1-1
System Overview 1-1
Basic Operation 1-2
Formatting Options 1-2
Requests 1-3
Default Report Elements 1-4
Page Layout and Titles 1-4
Separators 1-4
Folding and Width 1-4
Alignment 1-5
Optional Report Elements 1-5
Editing 1-6
Headers/Footers 1-6
Column Titles 1-7
Active Requests 1-7
Page Breaks 1-7
Excluding Columns 1-7
Ordering of Columns 1-7
Grouping 1-8
Outlining 1-8
Totals and Subtotals 1-8
Counts and Subcounts 1-8
Separators and Delimiters 1-8
Embedded Control Lines and Hyphenation 1-9
Full Page Formatting 1-9

Section 2 Report Writer Tutorial 2-1
General Report Options-1 2-4
Specific Column Options 2-6
General Report Options-2 2-16
Special Editing of a Report 2-24
Saving a Report and Resetting Options 2-27
Restoring a Saved Report 2-28
General Column Options 2-30

Section 3 Subroutine Overview and Tutorial 3-1
Creating An Invocation 3-1
Data Table Retrieval 3-3
Data Tables 3-6
Defining A Table 3-6
Row Information Structure 3-6
Subsystem Table Manager Procedure 3-7
The Row Value Buffer 3-7
create_table Entrypoint 3-8

get_row Entrypoint 3-8
delete_table Entrypoint 3-9
get_query Entrypoint 3-9
Data Conversions 3-10
Report Preparation 3-11
Report Formatting 3-11
Destroying An Invocation 3-11

Section 4 Report Writer Request Descriptions 4-1
column_value, clv 4-2
display, di 4-3
display_builtins, dib 4-8
list_format_options, lsfo 4-9
restore_format_options, rsfo 4-13
save_format_options, svfo 4-13
set_format_options, sfo 4-15

Section 5 Report Writer Subroutine Description 5-1
report_writer_ 5-2
convert_and_move_row 5-2
create_invocation 5-3
define_columns 5-4
destroy_invocation 5-7
set_report_writer_info_ptr 5-7
set_table_manager 5-8

Section 6 Table Manager Subroutine Description 6-1
table_manager$create_table 6-2
table_manager&delete_table 6-2
table_manager$get_query 6-3
table_manager$get_row 6-4

Section 7 Display Employee PL/1 Example 7-1
Main Procedure 7-1
Table Manager 7-2
create_table entry 7-2
delete_table entry 7-2
get_row entry 7-2
Internal Procedures 7-4
create_invocation 7-4
destroy_invocation 7-6
initialize 7-7
Declarations 7-8

Index i-1

ILLUSTRATIONS

Figure 3-1 Creating an Invocation 3-2

Figure 3-2 Selecting A Table 3-4
Figure 3-3 Displaying the Data 3-5
Figure 3-4 Destroying an Invocation 3-12

SECTION 1

REPORT WRITER SYSTEM

The Multics Report Writer (MRW) system is a generalized report writing facility that is included in a number of application subsystems. Utilizing this facility a user can:

1. create formatted reports
2. change and examine report layouts
3. save report layouts
4. restore report layouts

The MRW provides an end-user-oriented subsystem request interface, and a programmer subroutine interface. Facilities are provided for application subsystems to use the terminal-oriented end-user interface to control the report writing process entirely from a program, or to use a combination of the two interfaces.

SYSTEM OVERVIEW

The MRW produces formatted reports from data that is viewed as a table. Through this facility the user can control:

- page width and length
- page breaks
- page, group, and row headers/footers
- counts, subcounts, totals, and subtotals
- hyphenation of overlength values
- reordering and excluding selected columns
- duplicate suppression
- column alignment, editing, folding, separators, titles, and widths
- sorting on one or more columns
- directing the report to the terminal, a file, or an io switch
- horizontal and vertical scrolling through the report

The MRW is designed to serve the needs of the casual and experienced user. A casual user can have a default report layout provided by the system, while an experienced user can precisely define the report layout.

Basic Operation

The MRW retrieves rows of information from a table and produces a formatted output report. The rows retrieved are specified through a mechanism provided by the subsystem which uses the report writer (e.g., in the LINUS subsystem, the input_query and translate_query requests are used to select the table of interest).

Formatting Options

A formatted report is produced under the control of "formatting options." Formatting options consist of a name (for identity) and a set value. An example of a formatting option is:

```
-page_width 80
```

where -page_width is the name of this option and "80" is the set value associated with the name. Formatting options which deal with columns require an "option identifier" to uniquely identify the column. For example, to set the width of a column, an identifier is needed to determine which column the width is to be set for. Identifiers can be given as the number of the column in the query, the name of the column, or a star name which is matched against the column names. Examples of formatting options with identifiers are:

```
-width salary 10
-folding 3 fill
-alignment ** center
```

The formatting options are grouped into the following classifications:

general report options
: control the overall characteristics of a report. They are assigned default values when the application subsystem is first invoked, but can be changed by the user at any time. These values are retained for the entire session. General report options consist of:

```
-delimiter
-format_document_controls
-hyphenation
-page_footer_value
-page_header_value
-page_length
-page_width
-title_line
-truncation
```

general column options
: control the overall characteristics of the columns, such as examining the value of certain columns to determine if a page break is to be generated. They are assigned default values for every new query, but can be changed by the user at any time. These values are retained only during the current query (i.e., until the next new query is generated). General column options consist of:

```
-column_order
-count
-exclude
-group
-group_footer_trigger
-group_footer_value
-group_header_trigger
```

```
-group_header_value
-outline
-page_break
-row_footer_value
-row_header_value
-subcount
-subtotal
-total
```

specific column options
: control the characteristics of one specific column. They are assigned default values for every new query, but can be changed by the user at any time. These values are also retained only during the current query (i.e., until the next new query). These formatting options require an identifier to determine which column the particular option applies to. Specific column options consist of:

```
-alignment
-editing
-folding
-separator
-title
-width
```

The values of formatting options are listed and set through the list_format_options and set_format_options requests. These requests take control arguments which are the names of the formatting options. For example, to determine the current page width, enter:

```
list_format_options -page_width
```

and to change page width, enter:

```
set_format_options -page_width 71
```

A concept of "active" options is employed to make the system easier to use and to provide flexibility. For example, if a novice user does not set page headers, then no reference is made to them. If a user defines a page header, it then becomes active and appears in the output of the various reporting requests. If a user decides to eliminate a previously set page header, that is, by invoking the "set_format_options -page_header_value -default", it reverts back to the "inactive" state. This concept reduces the number of options listed when the user invokes the list_format_options request with no control arguments. The page_header_value is not listed if set to its default value as previously described.

Specific column options are active at all times, whereas general column options and general report options are active only when their value is set different from the original default value. For example, if the page_width is assigned its default value by the system, or is reverted to by the user, it is not active. The moment that it is changed to a value different from its default, it is considered active.

Requests

A number of requests are available for use in the creation of reports. Following is a brief summary of the report requests (refer to Section 4 for a detailed discussion of all requests):

column_value
: returns the value of the specified column for the current row, previous row, or the next row.

display
retrieves selected data, creates a report, and displays the information or writes it to a file or an io switch.

display_builtins
returns the current values for requested built-ins.

list_format_options
displays the names and values of formatting options.

restore_format_options
restores saved report layouts.

save_format_options
saves current values of formatting options for future use.

set_format_options
changes/sets report formatting options.

DEFAULT REPORT ELEMENTS

Page Layout and Titles

A page consists of a title line followed by as many rows as fit on the remainder of the page. The default title line is made up of one or more column titles, one column title for each column on the page. The column title is the column name as selected through the application subsystem. The row is made up of one or more columns, all concatenated together to form the row. The page width is 79 character positions and the page length is 66 lines, with 3 of these lines, at the top and bottom, reserved for margins.

Separators

A separator is provided for each column value and each column title. The default separator is two blanks placed between each pair of column titles and column values. The last column title or column value of a row has no separator.

Folding and Width

If when formatting a report, report elements do not fit within the defined width, "folding" takes place. Folding can occur in two different ways: "truncation" and "filling". Truncation means that the value is truncated to the defined width and the last displayable character is replaced by the truncation character(s) (normally "*"). Filling means that portions of the value are moved down to the next line(s), allowing the newly formatted value to appear within its defined width. The format_document_ subroutine (described in the Subroutines Manual) is used to provide filling of overlength values, and format document controls can optionally be supplied to provide greater control over the filling action. Filling takes place when a value is wider than its display width; when the value contains vertical tabs characters, horizontal tab characters, backspace characters, or newline characters; or when the alignment mode is set to "both". When column values do not have editing requests associated with them, the value is trimmed first (i.e., before the test for filling is done). Character and bit data types have trailing blanks trimmed, and all other data types have leading and trailing blanks trimmed.

The default width for a column value is derived from the columns selected through the application subsystem. The width chosen is the exact number of characters needed to contain the value after it is converted from the subsystems internal data type, to character format, via PL/I conversion rules. When the default width is used, the column value always fits, but this width can be reduced by the user. The reduction of the column width can cause folding to occur. Column folding can be set to "fill" or "truncate" and proceeds as described above. The default for column values is "fill."

The concatenation of all column values and separators (used to determine row value) can cause row folding to occur. This happens when the resulting row is wider than the defined page width. In this case, columns which appear on or to the right of the right page boundary are moved down to the next line(s). The corresponding titles are moved so that they appear directly over the columns. Columns whose widths are greater than the page width are automatically reduced to the page width.

Alignment

The alignment for column values is derived from the data type of the column, as defined by the application subsystem that uses the MRW. Character and bit strings default to "left-alignment," decimal data with a non-zero scale defaults to "decimal-point-alignment," and all other data types default to "right-alignment." The user can set the alignment of individual columns to left, right, center, both, or decimal-point-alignment.

The alignment for a column title is center (i.e., the title is centered within its defined width).

The alignment for a title line or a row is left (i.e., the title line or row is placed against the left page boundary).

OPTIONAL REPORT ELEMENTS

A number of optional features (for greater control over report appearance) are available for more sophisticated report formatting. These optional features are:

- editing
- headers/footers
- column titles
- active requests
- page breaks
- excluding columns
- ordering of columns
- grouping
- outlining
- totals and subtotals
- counts and subcounts

- separators and delimiters
- embedded control lines and hyphenation

Editing

Editing can be specified for any column value, and is provided by subsystem active requests and Multics active functions. The report column_value request is used to pass the value to other active requests, and the returned value is then folded and aligned as described above (see "Folding and Alignment"). The MRW does not strip a level of quotes from the editing request; the first time quote-stripping occurs is when ssu_$evaluate_active_string subsystem utilities procedure is invoked. Editing of column values is not provided by default.

Headers/Footers

A header or footer is a character string provided by the user. The character string can contain active requests, be made up of more than one "portion," and consist of more than one line. A delimiter character is used to separate the different portions of a header or footer. The delimiter character default is "!", but can be changed by the user. The header/footer can consist of a left, right, and center page portion.

Evaluation of a header/footer is a two-part operation that proceeds in the following manner: first, the header/footer is divided into its portions based on the delimiter character; and second, active requests are evaluated. Quote-stripping is not done by the MRW during these two operations; the first time quote-stripping occurs is when the ssu_$evaluate_active_string subsystem utilities procedure is invoked. The MRW display_builtins active request can be used to obtain built-ins like the current page number in a header/footer, and the MRW column_value active request can be used to obtain the value of a column.

A header or footer can be made up of a left, right, and center page portion. These portions are determined by the delimiter characters. The portions are aligned to the left, right, and center of the page. Folding on headers/footers proceeds independently for each part. Portions of a header or footer (left, right, or center) with zero length are redistributed to other portions whose lengths are not zero. For example, if the page header contained only a center portion as:

```
!!Sample Center Portion!!
```

the text would be centered on the page, but would have the full page width available for the text. Similarly, a left portion or right portion only is aligned to the left or right of the page, but has the full page width available for placement of its text. Two exceptions to this action are when the header or footer has a left, right, and center portion, and the left or right portion has a zero length. For example:

```
!left part!center part!!
```

or

```
!!center part!right part!
```

In both cases the left or right part of the page is unavailable for placement of text (i.e., the space is not redistributed to the other two portions).

If redistribution of the available page width is not desired, the placement of a single blank into a portion prevents the redistribution from taking place because the portion has a length greater than zero. For example:

```
!  !Center Part!  !
```

Headers and footers can be defined for a page, group, and a row. The first row that appears on the page is available for the page header, and the last row that appears on the page is available for the page footer. The first row of a group is available for the group header, and the last row of a group is available for the group footer. The current row is available for use in the row header and row footer.

Column Titles

A column title is a character string that is placed above its associated column. The display width available for the title is inherited from its parent column, along with the folding action. If the title is exactly the same number of characters as the display width, it is placed without any folding or alignment action. If the title is shorter, it is centered within the display width. If the title is wider, it is truncated or filled, depending on its parent column's setting.

Active Requests

Active requests are used in headers/footers to substitute values into the header/footer at the time the report is being formatted. For example, the Multics date active function can be used to provide the current date as part of the header or footer.

Active requests are also used to provide editing for column values which become part of the row value. For example, the subsystem execute request and the Multics picture active function can be used to provide editing features such as dollar signs and commas.

The user specifies subsystem active requests through the construct "[name STR]", where name is the name of the desired active request and STR is any argument(s) required by the active request. Multics active functions are invoked via the subsystems [execute] active request. They are specified by the user through the construct [execute name STR], where name is the name of the Multics active function and STR is any argument(s) required by the active function. The active function/request is evaluated and its returned value is substituted into the original string before folding and alignment take place.

Page Breaks

Page breaks can be set to occur when the value of one or more columns change. The occurrence of a new value in the column(s) being examined closes out the current page and a new page is started. The new row which caused the page break is not made available until the start of the next page. This allows the page footer to access the correct row (the last row on that page).

Excluding Columns

Columns selected through the subsystem can be excluded from the row value. Through use of the [column_value] active request, the column value can be obtained for placement elsewhere on the page. For example, a user may exclude the display of a column that is being used to determine when to generate page breaks, and place the value of the column in the page header with the column_value active request.

Ordering of Columns

Columns appear on the page in the order they were selected through the subsystem. This order can be changed by the user without having to go back and select a different order through the subsystem.

Grouping

One or more columns can be used to define a "group" of rows based on the values of these columns. The named columns make up a major to minor hierarchy and can be used in conjunction with the outlining, page break, subtotal, and subcount features.

Outlining

One or more columns can have duplicate values suppressed. If the value of the current column is the same as the previous value, then its display is suppressed unless it is the first line on a new page.

If any named column is a member of the group of columns defined via the grouping feature, it and any columns more major in the hierarchy are outlined. A change in value of any one column displays all values of columns lower in the hierarchy in addition to the changed column. An exception is the first line on a new page, when duplicate values are never suppressed.

Totals and Subtotals

Totals and subtotals can be specified for columns. The totals and subtotals are placed directly under the associated columns.

A column subtotal is generated when the value of the column(s) the subtotal is associated with changes. The subtotal can be associated with one or more columns. Several subtotals can be specified, each associated with different columns. Subtotals can be "reset" or "running." A column total is generated after the last input row is processed.

The width, alignment, folding, and editing request for a total or subtotal is inherited from its parent column. During the generation of a total or subtotal, the column_value request returns the value of the total or subtotal, rather than the column value. When the parent column is excluded from the page, the total or subtotal associated with it is also excluded. An exception to this rule is when all of the columns have been excluded. They are provided in this case to produce reports containing some combination of subcounts, subtotals, counts, and totals only.

Counts and Subcounts

Counts and subcounts can be specified for columns, and function as described above under "Totals and Subtotals." A count or subcount counts occurrences of values, whereas a total or subtotal accumulates values.

Separators and Delimiters

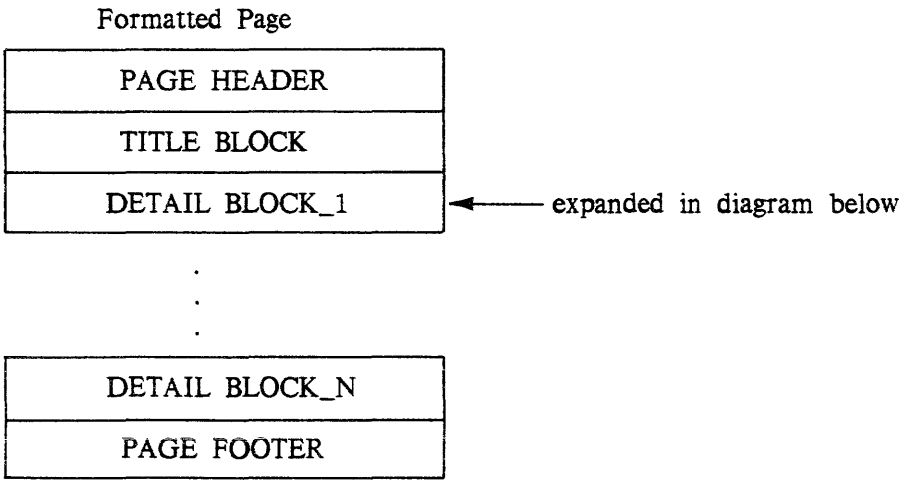
The separators used to separate column values and column titles from each other can be set to any string of displayable characters by the user. The delimiter character used to delimit the different portions of a header/footer can also be set by the user.

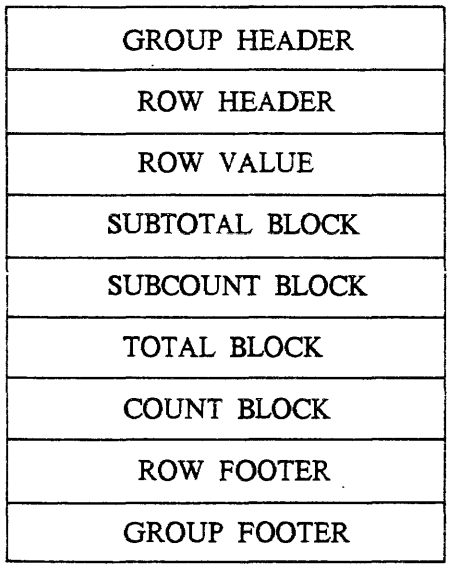
Embedded Control Lines and Hyphenation

The MRW uses the format_document_ subroutine (refer to the Subroutines Manual) to "fill" overlength text. A user can embed format document control lines in text to achieve greater control of the filling action. A user can also specify that hyphenation of words should be attempted when filling overlength text.

FULL PAGE FORMATTING

The MRW system formats a full page before any output is provided. It operates in this fashion because it is sometimes necessary to back up on a page and defer report elements to the next page so that associated report elements remain on the same page. A full page with all report elements present is outlined in the following diagram.





All of the defined report elements are optional, but at least one must be present or a zero-length page results. A zero-length page is treated as an error and the report formatting is terminated.

Backing up on a page is accomplished through a detection and prevention method, and proceeds as follows:

1. The page header, if present, is processed first. If the page header does not fit on the page, it is treated as an error and the report formatting is terminated. The formatted page header can fill the complete page if no other report elements are defined.
2. The title line, if present, is processed next. If the title line does not fit on the page, it is treated as an error and the report formatting is terminated. The formatted title block can fill the complete page if no other report elements are defined.
3. The detail block is processed next. A detail block can be made up of a group header, a row header, a row value, a subtotal block, a subcount block, a total block, a count block, a row footer, and a group footer. These different elements are treated as one unit and must all appear on one page or the detail block is deferred to the next page. If any of these elements are defined, then at least one detail block must fit on the page or it is treated as an error and the report formatting is terminated. The formatted detail block can fill the complete page if no other report elements are defined.
    a. The group header, if present, is processed first. If the current row is the first row of the report, or if the column associated with the -group_header_trigger option has just changed with the current row, the header is generated. If the group header does not fit on the page, the detail block is deferred to the next page, provided one detail block is already placed on the page.
    b. The row header, if present, is processed next. If the row header does not fit on the page, the detail block is deferred to the next page, provided one detail block is already placed on the page.
    c. The row value, if present, is processed next. If the row value does not fit on the page, the detail block is deferred to the next page, provided one detail block is already placed on the page. The editing requests associated with any columns are evaluated before an attempt is made to place the row value on the page. If the row value is deferred to the next page for any reason, the editing requests associated with the columns are evaluated again when the row value is processed on the next page. This is necessary to ensure that obtained values, such as the page_number display built-in are correct. For users who are doing calculations based on accumulations, this could produce incorrect calculations. That is, the value of a row could be accumulated more than once. The previously_processed_row display built-in provides a mechanism to ensure this does not happen. If the value of this built-in is true, a user doing accumulations would not add in the current row value as it was already added in when the editing requests for the row were processed the first time.
    d. The row subtotal, if present, is processed next. If subtotal generation is necessary, and the row subtotal does not fit on the page, the detail block is deferred to the next page, provided one detail block is already placed on the page. The editing requests associated with any subtotals are only evaluated when subtotal generation is done, and proceed as described above under "row value" editing requests evaluation. The previously_processed_row display built-in also functions as described above.

e. The row subcount, if present, is processed next. It proceeds as described above under row subtotal (item d).

f. The row total, if present, is processed next. If total generation is necessary, and the row total does not fit on the page, the detail block is deferred to the next page, provided one detail block is already placed on the page. The editing requests associated with any totals are only evaluated when total generation is done, and proceed as described above under "row value" editing requests evaluation. The previously_processed_row display built-in also functions as described above.

g. The row count, if present, is processed next. It proceeds as described above under row subtotal (item d).

h. The row footer, if present, is processed next. If the row footer does not fit on the page, the detail block is deferred to the next page, provided one detail block is already placed on the page.

i. The group footer, if present, is processed last. If the current row is the last row of the report, or the column associated with the -group_footer_trigger option is about to change with the next row, the footer is generated. If the group footer does not fit on the page, the detail block is deferred to the next page, provided one detail block is already placed on the page.

4. The page footer, if present, is processed last. If the page footer does not fit on the page, the last detail block on the page is removed and the page footer is processed again. Active requests found in the footer are evaluated again to ensure correct processing of display built-ins like current_row_number. If the page footer still does not fit, another detail block is removed from the page and the footer is evaluated again. This process continues until the footer fits, or there are no more detail blocks to remove from the page. The first detail block that appears on the page is never removed, and if its removal is necessary to provide a fit for the page footer, it is treated as an error and report formatting is terminated.

SECTION 2

REPORT WRITER TUTORIAL

This section consists of report writer examples organized into a sample user session. An application subsystem called "display_employee" was constructed for these examples, and a complete listing of it can be found in Section 6. User-typed lines and lines displayed by the system are shown together in the example. To differentiate between these lines, an exclamation mark (!) precedes user-typed text. This is done only to distinquish user text from system-generated text; it is not to be included as part of the input line. Also, a "carriage return" (moving the display mechanism to the first column of the next line, called a newline or NL on Multics) is implied at the end of every user-typed line. Line numbers are also included in the examples for purposes of commentary immediately following the example.

Note: Because of page constraints in this document, certain character strings of data used in examples may not match exactly the information as seen on a user's terminal. That is, the character strings in examples may be folded or multiple-lined, whereas the actual interactive (live) session may display the same information on a single line or multiple lines with different line breaks than shown here. Additionally, some blank lines have been removed in the examples for space consideration. In most cases this can be recognized by the reader. For example:

```
55 ! display_employee:  display -page 1

59    (system display)
```

Only one space is used to separate the lines, but the line numbers to the left of the lines imply there are actually three spaces here.

In other cases, lines have purposely been skipped or eliminated because of changes made while developing the examples.

Following is a list of request and control argument abbreviations used in the tutorial examples. They are included here for the purpose of saving the reader from referring to other sections of the manual if a term is unfamiliar. The list includes only those abbreviations used within this section.

REQUEST ABBREVIATIONS

clv	column_value
di	display
dib	display_builtins
e	execute
ec	exec_com
ls	list (Multics command level)
lsfo	list_format_options
pr	print (Multics command level)
q	quit
rsfo	restore_format_options

This page intentionally left blank.

sfo	set_format_options
svfo	save_format_options
w	write

CONTROL ARGUMENT ABBREVIATIONS

-al	-alignment
-co	-column_order
-dm	-delimiter
-ed	-editing
-ex	-exclude
-fdc	-format_document_controls
-fold	-folding
-gft	-group_footer_trigger
-gfv	-group_footer_value
-ght	-group_header_trigger
-ghv	-group_header_value
-gr	-group
-hph	-hyphenation
-kr	-keep_retrieval
-krp	-keep_report
-nr	-new_retrieval
-of	-output_file
-or	-old_retrieval
-orp	-old_report
-out	-outline
-pb	-page_break
-pfv	-page_footer_value
-pg	-page
-phv	-page_header_value
-pl	-page_length
-pw	-page_width
-rfv	-row_footer_value
-rhv	-row_header_value
-rs	-reset
-sep	-separator
-stt	-subtotal
-tc	-truncation
-td	-temp_dir
-tl	-title_line
-tt	-total
-ttl	-title
-wid	-width

GENERAL REPORT OPTIONS-1

```
 1 ! display_employee
 5 ! display_employee:  lsfo -dm -fdc -hph -pfv -phv -pl -pw -tl -tc
     -delimiter                    "!"
     -format_document_controls     "off"
     -hyphenation                  "off"
     -page_footer_value            ""
     -page_header_value            ""
     -page_length                  "0"
     -page_width                   "79"
     -title_line                   "on"
     -truncation                   "*"

 7 ! display_employee:  sfo -pw 0

 8 ! display_employee:  lsfo -pw
     -page_width                   "0"

 9 ! display_employee:  sfo -pw -default

10 ! display_employee:  lsfo -pw
     -page_width                   "79"
```

line 1
Invoke display_employee.

line 5
List the names and values of specified report formatting options. All of the displayed values in this case are "default" values. These options are the "general report options." They remain in effect across the entire subsystem session. For example, if the page width is changed, it remains at this new value until it is explicitly changed back, or until the subsystem session is terminated.

-dm "!"	character used to delimit portions of header/footer.
-fdc "off"	used when filling overlength character strings. If "off," ignore embedded controls.
-hyphenation "off"	used when filling overlength character strings. If "off," do not attempt to hyphenate words.
-pfv ""	footer placed at bottom of each page.
-phv ""	header placed at top of each page.
-pl "0"	length of each formatted page (number of lines).
-pw "79"	width of each formatted page (number of character positions).
-tl "on"	print the title line.
-tc "*"	character that indicates truncation has occurred.

lines 7-10
Set page width, list page width, reset page width, and list page width once again.

SPECIFIC COLUMN OPTIONS

The following example looks at "specific column options." These options are always listed and are assigned new default values each time a new set of columns are selected.

```
 1 ! display_employee:  lsfo
 6   -alignment age              "right"
     -alignment city             "left"
     -alignment family           "left"
     -alignment job              "right"
     -alignment name             "left"
     -alignment salary           "decimal 8"
     -alignment sex              "left"
     -alignment state            "left"
14   -editing age                ""
     -editing city               ""
     -editing family             ""
     -editing job                ""
     -editing name               ""
     -editing salary             ""
     -editing sex                ""
     -editing state              ""
22   -folding age                "fill"
     -folding city               "fill"
     -folding family             "fill"
     -folding job                "fill"
     -folding name               "fill"
     -folding salary             "fill"
     -folding sex                "fill"
     -folding state              "fill"
30   -separator age              "  "
     -separator city             "  "
     -separator family           "  "
     -separator job              "  "
     -separator name             "  "
     -separator salary           "  "
     -separator sex              "  "
     -separator state            "  "
38   -title age                  "age"
     -title city                 "city"
     -title family               "family"
     -title job                  "job"
     -title name                 "name"
     -title salary               "salary"
     -title sex                  "sex"
     -title state                "state"
46   -width age                  "5"
     -width city                 "13"
     -width family               "1"
     -width job                  "5"
     -width name                 "10"
     -width salary               "10"
```

```
     -width sex                    "1"
53   -width state                  "2"
```

---

line 1
List the names and values of the column options.

lines 6-13
System display -- the alignment option specifies how a value is to be aligned within its display width.

- Character and bit strings default to left-alignment.
- Decimal data with a non-zero scale defaults to decimal-point-alignment.
- All other data types default to right-alignment.

lines 14-21
System display -- the editing option provides additional editing for column values. (Default is no editing)

lines 22-29
System display -- the folding option specifies the action taken when the column value exceeds the display width for the column. (Default is fill)

lines 30-37
System display -- the separator option specifies the character string that separates the specified column from the following column. (Default is two blanks)

lines 38-45
System display -- the title option specifies the character string to be placed at the top of the page above the column. (Default is the column name)

lines 46-53
System display -- the width option specifies the display width of the detail line of the column. (Default is the number of characters needed after conversion to character format)

The following examples look at a report utilizing the available specific column options.

```
55 ! display_employee:  sfo -pl 66; di -pg 1

59      name       job      salary      age    s  f  st      city
                                              e  a  at
                                              x  m  e
                                                 i
                                                 l
                                                 y

66    abel             1    14555.01      36   m  s  ak   juneau
67    abell            2    13000.01      55   f  m  az   phoenix
68    abernathy        3    12500.01      61   m  d  ca   fresno
69    abodoura         5    12900.01      61   m  m  ca   sacramento
70    aboe             4    10201.01      41   f  s  ca   los angeles
71    abraham          6    15000.01      25   f  d  ca   san diego
72    abrahms          7    14300.01      35   m  s  ca   san francisco
      .
      . (45 data lines)
      .
118   baker            1    12000.10      71   m  s  il   springfield
```

line 55

Set page length to 66 and display page 1. Data is retrieved and formatted by default parameters.

> **Note:** Multiple report writer requests can be included in a single request line by utilizing the request termination character (;) between requests. Any number of requests may be included on a line using this format.

lines 59-118

System display

```
120 ! display_employee:  lsfo -wid state
      -width state                "2"

123 ! display_employee:  sfo -wid state 5
125 ! display_employee:  di -pg 1

129      name       job     salary      age   s  f  state      city
                                             e  a
                                             x  m
                                                i
                                                l
                                                y

136   abel            1    14555.01      36   m  s  ak     juneau
137   abell           2    13000.01      55   f  m  az     phoenix
138   abernathy       3    12500.01      61   m  d  ca     fresno
139   abodoura        5    12900.01      61   m  m  ca     sacramento
140   aboe            4    10201.01      41   f  s  ca     los angeles
141   abraham         6    15000.01      25   f  d  ca     san diego
142   abrahms         7    14300.01      35   m  s  ca     san francisco
      .
      . (45 data lines)
      .
188   baker           1    12000.10      71   m  s  il     springfield
```

---

line 120
List the width value of the "state" column.

lines 123–125
Set the width value for state column to "5" from its default value of "2", and display page 1.

lines 129–188
System display -- note the difference in the state column header on line 129 from that displayed on lines 59–61.

```
190 ! display_employee:  lsfo -wid 8
     -width city                "13"

193 ! display_employee:  sfo -wid 8 10
195 ! display_employee:  di -pg 1

199      name       job      salary      age   s  f  state      city
                                              e  a
                                              x  m
                                                 i
                                                 l
                                                 y

206   abel             1     14555.01      36   m  s  ak       juneau
207   abell            2     13000.01      55   f  m  az       phoenix
208   abernathy        3     12500.01      61   m  d  ca       fresno
209   abodoura         5     12900.01      61   m  m  ca       sacramento
210   aboe             4     10201.01      41   f  s  ca       los
211                                                            angeles
212   abraham          6     15000.01      25   f  d  ca       san diego
213   abrahms          7     14300.01      35   m  s  ca       san
214                                                            francisco
      .
      .  (37 data lines)
      .
252   arnold          22     18210.01      53   f  d  pa       philadelph
253                                                            ia
254   ashman          23     12400.01      52   m  s  tn       chattanoog
255                                                            a
256   ashworth        24      9301.01      61   f  m  tx       austin
257   asin             1     15100.01      51   m  d  tx       dallas
258   auburn           2     13101.01      70   f  s  vt       rutland
```

---

line 190
List the width value of column 8 (city)

lines 193–195
Set the width value of the 8th column to "10" from its default value of "13", and display page 1.

lines 199–258
System display -- note the difference under the city header (lines 210–214) from that displayed on lines 140–142. Also notice the not-so-pleasant breakup of lines 252–255. This is an example of column "filling."

```
260 ! display_employee:  sfo -wid 8 -default;lsfo -wid name
      -width name                 "10"

263 ! display_employee:  sfo -wid name 7 -fold name truncate
265 ! display_employee:  di -pg 1

269      name      job     salary     age   s  f  state     city
                                           e  a
                                           x  m
                                              i
                                              l
                                              y

277   abell        2    13000.01     55  f  m  az     phoenix
278   aberna*      3    12500.01     61  m  d  ca     fresno
279   abodou*      5    12900.01     61  m  m  ca     sacramento
280   aboe         4    10201.01     41  f  s  ca     los angeles
      .
      .  (47 data lines)
      .
328   baker        1    12000.10     71  m  s  il     springfield
```

---

line 260
Set the width value of column 8 (city) to its default value (13) and list the width value of the name column.

lines 263–265
Set the width value of the name column to "7", truncate the data listed under the name column, and display page 1.

lines 269–279
System display -- note the difference under name header (lines 278–279) from that displayed on lines 208–209.

```
330 ! sfo -sep ** " | "
332 ! display_employee:  di -pg 1

336    name    |  job  |   salary    |  age  | s | f | state |     city
               |       |             |       | e | a |       |
               |       |             |       | x | m |       |
               |       |             |       |   | i |       |
               |       |             |       |   | l |       |
               |       |             |       |   | y |       |

343   abel     |     1 |   14555.01  |    36 | m | s | ak    | juneau
344   abell    |     2 |   13000.01  |    55 | f | m | az    | phoenix
345   aberna*  |     3 |   12500.01  |    61 | m | d | ca    | fresno
346   abodou*  |     5 |   12900.01  |    61 | m | m | ca    | sacramento
347   aboe     |     4 |   10201.01  |    41 | f | s | ca    | los angeles
      .
      .  (47 data lines)
      .
395   baker    |     1 |   12000.10  |    71 | m | s | il    | springfield
```

lines 330–332
Set the column separator value to "<SP>|<SP>" from its default value of <SP><SP> (two blanks), and display page 1.

lines 336–395
System display -- note that the columns have shifted to the right because the separator was increased to three character positions. Previous example separators were only two character positions.

```
397 ! display_employee:  sfo -al age left -ed salary -prompt
398   Enter -editing salary.
399 ! [e pic $zz,zz9v.99 [clv salary]]
400 ! .

402 ! display_employee:  di -pg 1

406    name    |  job  |    salary    |  age  | s | f | state |     city
               |       |              |       | e | a |       |
               |       |              |       | x | m |       |
               |       |              |       |   | i |       |
               |       |              |       |   | l |       |
               |       |              |       |   | y |       |

413    abel    |     1 | $14,555.01 | 36    | m | s | ak    | juneau
       abell   |     2 | $13,000.01 | 55    | f | m | az    | phoenix
       aberna* |     3 | $12,500.01 | 61    | m | d | ca    | fresno
       abodou* |     5 | $12,900.01 | 61    | m | m | ca    | sacramento
       .
       .  (48 data lines)
       .
464    baker   |     1 | $12,000.10 | 71    | m | s | il    | springfield

                ---------------------------------
```

line 397
: Set alignment value for age column to "left" from its default value of "right," and invoke the editing option with prompt.

lines 399–402
: Edit request, termination, and display page 1.

lines 406–464
: System display -- note that the information under the age column is now aligned to the left of the column and the data under the salary column contains the "$" and "," characters.

```
466 ! display_employee:  lsfo -ttl **
      -title age                  "age"
      -title city                 "city"
      -title family               "family"
      -title job                  "job"
      -title name                 "name"
      -title salary               "salary"
      -title sex                  "sex"
      -title state                "state"

476 ! display_employee:  sfo -al age -default;sfo -ttl (1 2 3 4 5 6 7 8) -prompt
477   Enter -title name.
478 ! NAME
479 ! .
480   Enter -title job.
481 ! JOB
482 ! .
483   Enter -title salary.
484 ! SALARY
485 ! .
486   Enter -title age.
487 ! AGE
488 ! .
489   Enter -title sex.
490 ! SEX
491 ! .
492   Enter -title family.
493 ! FAMILY
494 ! .
495   Enter -title state.
496 ! STATE
497 ! .
498   Enter -title city.
499 ! CITY
500 ! .

502 ! display_employee:  di -pg 1

506    NAME    |  JOB  |    SALARY    |  AGE  | S | F | STATE |     CITY
              |       |              |       | E | A |       |
              |       |              |       | X | M |       |
              |       |              |       |   | I |       |
              |       |              |       |   | L |       |
              |       |              |       |   | Y |       |

      abel     |     1 | $14,555.01 |    36 | m | s | ak    | juneau
      abell    |     2 | $13,000.01 |    55 | f | m | az    | phoenix
      aberna*  |     3 | $12,500.01 |    61 | m | d | ca    | fresno
      .
      .  (49 data lines)
      .
565   baker    |     1 | $12,000.10 |    71 | m | s | il    | springfield
```

line 466
List the title values of all columns.

lines 476–502
Set the title value for all columns to new values (in this case, all have been changed from lowercase to uppercase), and display page 1.

lines 506–565
System display -- note that the column header values on line 506 are different from that displayed on line 406.

GENERAL REPORT OPTIONS-2

The following examples look at the "general report options" and are an extension to the example shown above under "General Report Options-1."

```
 1 ! display_employee:  sfo -pl 26
 3 ! display_employee:  di -pg 1

 7    NAME    |  JOB  |   SALARY    |  AGE  | S | F | STATE |     CITY
              |       |             |       | E | A |       |
              |       |             |       | X | M |       |
              |       |             |       |   | I |       |
              |       |             |       |   | L |       |
              |       |             |       |   | Y |       |

      abel    |     1 | $14,555.01  |    36 | m | s | ak    | juneau
      abell   |     2 | $13,000.01  |    55 | f | m | az    | phoenix
      aberna* |     3 | $12,500.01  |    61 | m | d | ca    | fresno
      .
      .  (9 data lines)
      .
27    agee    |    14 | $30,900.01  |    70 | m | s | hi    | honolulu

                ---------------------------------
```

lines 1-3
: Set the page length value to "26" from its previously set value of "66" (see line 55 above), and display page 1.

lines 7-27
: System display -- note that the report length has decreased. There are now 20 text lines and three "margin lines" at the top and at the bottom of the page. When the report is sent to a file (for later printing), these six margin lines are put in the report by the line printer software. This produces the same page format, whether viewed at a terminal or on hardcopy.

```
29 ! display_employee:  sfo -phv -prompt
30   Enter -page_header_value.
31 ! ![e date]!Sample Report![e time]!
32 ! !!!!
33 ! .

35 ! display_employee:  di -pg 1

39   04/29/83                          Sample Report                          10.26

      NAME    |  JOB  |   SALARY    |  AGE  | S | F | STATE |     CITY
              |       |             |       | E | A |       |
              |       |             |       | X | M |       |
              |       |             |       |   | I |       |
              |       |             |       |   | L |       |
              |       |             |       |   | Y |       |

      abel    |     1 | $14,555.01 |    36 | m | s | ak    | juneau
      abell   |     2 | $13,000.01 |    55 | f | m | az    | phoenix
      aberna* |     3 | $12,500.01 |    61 | m | d | ca    | fresno
      .
      .  (7 data lines)
      .
58    adkins  |    11 | $20,700.01 |    75 | m | m | fl    | key west

                ----------------------------------
```

line 29
: Set the page header value when prompted by the system.

line 30
: System display -- prompt

lines 31-35
: Set page header to contents of lines 31-32 (two header lines), terminate, and display page 1.

lines 39-58
: System display -- note that a page header (line 39) is now included as part of the report. This two-line page header reduces the page content of the report (i.e., the report now consists of 18 data lines whereas the previous example contained 20 lines). The page header fills the entire page width, but the column values do not. If the page width is set to zero, the display request calculates the page width to be an exact fit (i.e., contains all of the column values and separators).

```
60 ! display_employee:  sfo -pw 0
62 ! display_employee:  di -pg 1

66   04/29/83                    Sample Report                          10:26

      NAME    |  JOB  |   SALARY    |  AGE  | S | F | STATE |     CITY
              |       |             |       | E | A |       |
              |       |             |       | X | M |       |
              |       |             |       |   | I |       |
              |       |             |       |   | L |       |
              |       |             |       |   | Y |       |

     abel     |     1 | $14,555.01  |    36 | m | s | ak    | juneau
     abell    |     2 | $13,000.01  |    55 | f | m | az    | phoenix
     aberna*  |     3 | $12,500.01  |    61 | m | d | ca    | fresno
     .
     .  (7 data lines)
     .
85   adkins   |    11 | $20,700.01  |    75 | m | m | fl    | key west

              ---------------------------------
```

lines 60-62
Set the page width value to "0" from its default of "79," and display page 1.

lines 66,85
System display -- note that the page header is now centered over the columns. Setting the page width to zero has one disadvantage: when set to some positive integer and a column width exceeds the page width, that column width is reduced to the page width. For example, if the page width is set to 80 and the width for a column is set to 1024, the column width is reduced by the display request to 80. The reduction of a column display width does not take place when the page width is set to zero.

```
87 ! display_employee:  sfo -pfv -prompt
88   Enter -page_footer_value.
89 ! !!!!
90 ! !!- Page [dib page_number] -!!
91 ! .

93 ! display_employee:  di -pg 1

97   04/29/83                    Sample Report                        10:26

      NAME    |  JOB  |   SALARY    |  AGE  | S | F | STATE |     CITY
              |       |             |       | E | A |       |
              |       |             |       | X | M |       |
              |       |             |       |   | I |       |
              |       |             |       |   | L |       |
              |       |             |       |   | Y |       |

     abel     |     1 | $14,555.01 |    36 | m | s | ak    | juneau
     abell    |     2 | $13,000.01 |    55 | f | m | az    | phoenix
     aberna*  |     3 | $12,500.01 |    61 | m | d | ca    | fresno
     abodou*  |     5 | $12,900.01 |    61 | m | m | ca    | sacramento
     aboe     |     4 | $10,201.01 |    41 | f | s | ca    | los angeles
     abraham  |     6 | $15,000.01 |    25 | f | d | ca    | san diego
     abrahms  |     7 | $14,300.01 |    35 | m | s | ca    | san francisco
     acee     |     8 | $12,700.01 |    34 | f | m | co    | denver
114  acord    |     9 | $10,500.01 |    41 | m | d | ct    | hartford

116                                  - Page 1 -
```

---

line 87
Set the page footer value when prompted by the system.

line 88
System display -- prompt

lines 89–93
Set the page footer to contents of lines 89–90 (two footer lines), terminate, and display page 1.

lines 97–116
System display -- note that a page footer (line 116) is now included as part of the report. This two-line page footer reduces the page content of the report by another two lines (now 16 lines of data between header and footer).

```
118 ! display_employee:  sfo -pl 66;di -pg 1

122   04/29/83                    Sample Report                          10:26

       NAME    |  JOB  |   SALARY    |  AGE  | S | F | STATE |     CITY
               |       |             |       | E | A |       |
               |       |             |       | X | M |       |
               |       |             |       |   | I |       |
               |       |             |       |   | L |       |
               |       |             |       |   | Y |       |

       abel    |     1 | $14,555.01 |    36 | m | s | ak    | juneau
       abell   |     2 | $13,000.01 |    55 | f | m | az    | phoenix
       aberna* |     3 | $12,500.01 |    61 | m | d | ca    | fresno
       .
       .  (45 data lines)
       .
179    azer    |     5 | $12,600.01 |    44 | m | s | va    | norfork

181                                - Page 1 -
```

line 118
: Set the page length to 66 lines from its previous setting of "26" (see line 1 of this example set).

lines 122–181
: System display — note that the page now consists of 66 lines (3 blank margin lines at top and bottom and 60 lines of report).

```
183 ! display_employee:  sfo -tc <MORE>;di -pg 1

187   04/29/83                    Sample Report                          10:26

189    NAME    |  JOB  |   SALARY    |  AGE  | S | F | STATE |     CITY
               |       |             |       | E | A |       |
               |       |             |       | X | M |       |
               |       |             |       |   | I |       |
               |       |             |       |   | L |       |
194            |       |             |       |   | Y |       |

       abel    |     1 | $14,555.01  |    36 | m | s | ak    | juneau
       abell   |     2 | $13,000.01  |    55 | f | m | az    | phoenix
198    a<MORE> |     3 | $12,500.01  |    61 | m | d | ca    | fresno
199    a<MORE> |     5 | $12,900.01  |    61 | m | m | ca    | sacramento
       aboe    |     4 | $10,201.01  |    41 | f | s | ca    | los angeles
       .
       .  (43 data lines)
       .
244    azer    |     5 | $12,600.01  |    44 | m | s | va    | norfork

246                              - Page 1 -
```

---

line 183

Set the truncation value to "<MORE>" from its previous default value of "*," and display page 1. Refer to line 263 in the "Specific Column Options" example (above) where the width value of the name column was set to "7" and the folding option, with truncation (Default = *), was turned on for the name column.

lines 187-246

System display -- note the different truncation of the name column values (lines 198-199) from that displayed in the earlier example identified above (lines 278-279).

This page intentionally left blank.

```
248 ! display_employee:  sfo -tl off;di -pg 3

252   04/29/83                    Sample Report                          10:26

254   c<MORE> |     3 | $12,501.01 |    76 | m | m | ca    | san francisco
      cummins |     4 | $10,100.01 |    78 | f | d | co    | denver
      cutchin |     5 | $12,600.01 |    62 | m | s | ct    | hartford
      .
      .  (52 data lines)
      .
309   goodwyn |    15 | $12,400.01 |    39 | f | d | ct    | hartford

311                                - Page 3 -


313 ! display_employee:  sfo -tl on
```

---

line 248
: Set the title line value to "off" from its previous default value of "on," and this time display page 3. Turning the title line off inhibits the column header or title display from that displayed in the previous example (lines 189–194).

line 313
: Set the title line value to "on." This restores the display of column header or title lines.

Special Editing of a Report

The following example shows how to utilize a user-defined exec_com and interact with the editing request.

```
 1 ! display_employee:  sfo -wid sex 6 -ed sex "[ec sex_lookup [clv sex]]"
 3 ! display_employee:  ..ted
 4 ! a
 5 ! &version 2
 6 ! &trace off
 7 ! &if &[e equal m &1]
 8 ! &then &return male
 9 ! &else &return female
10 !  \f
11 ! w sex_lookup.display_employee
12 ! q

14 ! display_employee:  di -pg 1

18   04/29/83                        Sample Report                          10:26

20    NAME    |  JOB  |   SALARY    |  AGE  |   SEX    | F | STATE |     CITY
              |       |             |       |          | A |       |
              |       |             |       |          | M |       |
              |       |             |       |          | I |       |
              |       |             |       |          | L |       |
              |       |             |       |          | Y |       |

27   abel     |     1 | $14,555.01  |    36 | male     | s | ak    | juneau
28   abell    |     2 | $13,000.01  |    55 | female   | m | az    | phoenix
     a<MORE>  |     3 | $12,500.01  |    61 | male     | d | ca    | fresno
     .
     .  (45 data lines)
     .
75   azer     |     5 | $12,600.01  |    44 | male     | s | va    | norfork

77                                  - Page 1 -
```

line 1

Set the width of the sex column to "6" from its previous default value of "1," and prepare for special editing of the sex column data.

lines 3-12

Invoke the ted editor, append the following exec_com data (lines 5-9) into the ted buffer, terminate append mode, write the buffer to permanent storage, and quit the ted editor.

line 14

Display page 1.

lines 18-77

System display -- note the change in width of the sex column (line 20) from that displayed in the previous example (line 189) and the change of data by the exec_com (m = male and f = female).

This page intentionally left blank.

Saving a Report and Resetting Options

The following example shows how to save a report after it is in the desired format. Additionally, the example shows how to reset all options and revert the report back to its original format.

```
 1 ! display_employee:  svfo EXAMPLE-1.fo.display_employee;sfo -rs
 2 ! display_employee:  sfo -pl 66;di -pg 1

 5      name       job      salary      age    s  f  st       city
                                             e  a  at
                                             x  m  e
                                                i
                                                l
                                                y

        abel          1     14555.01      36  m  s  ak   juneau
        abell         2     13000.01      55  f  m  az   phoenix
        abernathy     3     12500.01      61  m  d  ca   fresno
        abodoura      5     12900.01      61  m  m  ca   sacramento
        aboe          4     10201.01      41  f  s  ca   los angeles
        abraham       6     15000.01      25  f  d  ca   san diego
        .
        .  (46 data lines)
        .
64      baker         1     12000.10      71  m  s  il   springfield
```

---

line 1
: Save the current values of format options as a subsystem exec_com (EXAMPLE-1.fo.display_employee) which can be restored later with the restore_format_options request. Then reset all options to their default values.

line 2
: Set page_length (-pl) to 66 lines, and display page 1.

lines 5-64
: System display — note that the report has reverted back to its original format (i.e., it is now the same as the first example in this sample user session).

At this point you may wish to terminate the display_employee session by entering:

```
65 ! display_employee:  q
66   (Multics ready message)
```

Restoring a Saved Report

The report saved in the previous example may be recalled at will. Assuming you want to have the report printed, then the following sequence of events must be set up:

```
3 ! display_employee
4 ! display_employee:  rsfo EXAMPLE-1
5 ! display_employee:  di -nr -pg 1

9   04/29/83                    Sample Report                          10:26

     NAME    |  JOB  |   SALARY    |  AGE  |  SEX    | F | STATE |     CITY
             |       |             |       |         | A |       |
             |       |             |       |         | M |       |
             |       |             |       |         | I |       |
             |       |             |       |         | L |       |
             |       |             |       |         | Y |       |

    abel     |     1 | $14,555.01 |    36 | male    | s | ak    | juneau
    abell    |     2 | $13,000.01 |    55 | female  | m | az    | phoenix
    a<MORE>  |     3 | $12,500.01 |    61 | male    | d | ca    | fresno
    a<MORE>  |     5 | $12,900.01 |    61 | male    | m | ca    | sacramento
    aboe     |     4 | $10,201.01 |    41 | female  | s | ca    | los angeles
    .
    .  (43 data lines)
    .
    azer     |     5 | $12,600.01 |    44 | male    | s | va    | norfork

69                                  - Page 1 -


71 ! display_employee:  di -of example-1
```

lines 3-4
Set up for restoring the saved format options.

line 5
Display page 1 of the report as a verification (i.e., is this the desired report?).

lines 9-69
System display -- note that the report is restored to its original condition (i.e., restored to the same format as that shown in the example under "Special Editing of a Report" above).

line 71
Write the complete formatted report to permanent storage in the user's working directory with pathname of "example-1".

The full report (example-1), along with the saved format options segment (EXAMPLE-1.fo.display_employee) now resides in the user's working directory and may be dprinted or retained in permanent storage at the user's discretion.

GENERAL COLUMN OPTIONS

The following examples look at the "general column options." These options remain in effect only for the duration of the current set of columns. Every time a new set of columns is selected, new default values are assigned. The options are listed (through use of the list_format_options request) when their value is different from the default, or when asked for by name.

```
1 ! display_employee:  lsfo -co
2   -column_order                "name job salary age sex family state city"

4 ! display_employee:  sfo -co 8 7 1 2 3 4 5 6;di -pg 1

8   04/29/83                      Sample Report                        10:28

        CITY       | STATE |  NAME    |  JOB  |   SALARY    |  AGE  |  SEX    | F
                   |       |          |       |             |       |         | A
                   |       |          |       |             |       |         | M
                   |       |          |       |             |       |         | I
                   |       |          |       |             |       |         | L
                   |       |          |       |             |       |         | Y

    juneau         | ak    | abel     |     1 | $14,555.01  |    36 | male    | s
    phoenix        | az    | abell    |     2 | $13,000.01  |    55 | female  | m
    fresno         | ca    | a<MORE>  |     3 | $12,500.01  |    61 | male    | d
    .
    .  (45 data lines)
    .
    norfork        | va    | azer     |     5 | $12,600.01  |    44 | male    | s

67                                  - Page 1 -
```

line 1
: List the current names and order of the report columns.

line 4
: Reorder the sequence of report columns, and display page 1.

lines 8–67
: System display -- note that the column order has been changed from that displayed in the previous example.

This page intentionally left blank.

```
74 ! display_employee:  lsfo -ex
75   -exclude                    ""

77 ! display_employee:  sfo -ex age job;di -pg 1

81   04/29/83              Sample Report                  10:31

     STATE |      CITY       |  NAME    |   SALARY    |  SEX    | F
           |                 |          |             |         | A
           |                 |          |             |         | M
           |                 |          |             |         | I
           |                 |          |             |         | L
           |                 |          |             |         | Y

     ak    | juneau          | abel     | $14,555.01 | male    | s
     az    | phoenix         | abell    | $13,000.01 | female  | m
     ca    | fresno          | a<MORE>  | $12,500.01 | male    | d
     .
     .  (45 data lines)
     .
     va    | norfork         | azer     | $12,600.01 | male    | s

140                          - Page 1 -
```

---

line 74
: List columns currently excluded from the report.

line 75
: System display -- the response is "", meaning that no columns are currently excluded.

line 77
: Exclude the age and job columns, and display page 1.

lines 81-140
: System display -- note that the age and job columns have been excluded from the report (i.e., the report now consists of six columns of data instead of the eight previously included).

```
142 ! display_employee:  sfo -ex "";lsfo -ex
143   -exclude                    ""
```

Execution of line 142 restores the age and job columns previously excluded by execution of line 77. Line 143 is the system display indicating that no columns are currently excluded.

The next few examples look at the "group" option which is used in conjunction with other requests. This option is used to define a "group" of rows based on the content of one or more columns.

```
145 ! display_employee:  lsfo -gr
146   -group                        ""

148 ! display_employee:  sfo -gr state city sex;lsfo -out
149   -outline                      ""

151 ! display_employee:  sfo -out sex;di -sort state city sex -pg 1,2

155   04/29/83                        Sample Report                          10:33

      STATE |      CITY      |  NAME   |  JOB  |   SALARY   |  AGE  |  SEX   | F
            |                |         |       |            |       |        | A
            |                |         |       |            |       |        | M
            |                |         |       |            |       |        | I
            |                |         |       |            |       |        | L
            |                |         |       |            |       |        | Y

      ak    | juneau         | bambry  |    10 | $11,501.01 |    66 | female | d
            |                | gaskins |     6 | $14,700.01 |    31 |        | s
            |                | justin  |     2 | $12,000.01 |    78 |        | m
      .
      .  (16 data lines)
      .
      az    | phoenix        | abell   |     2 | $13,000.01 |    55 | female | m
            |                | c<MORE> |    22 | $18,300.01 |    38 |        | d
            |                | june    |    18 | $10,900.01 |    73 |        | s
      .
      .  (12 data lines)
      .
            | tucson         | monaco  |    20 | $12,300.01 |    30 | female | d
            |                | nevitte |    15 | $12,300.01 |    77 |        | s
            |                | pauley  |    10 | $11,600.01 |    56 |        | m
201         |                | n<MORE> |     5 | $12,400.01 |    57 | male   | m
            |                | ordeman |     1 | $15,200.01 |    21 |        | d
      ca    | fresno         | bane    |    13 | $15,200.01 |    50 | female | m
      .
      .  (6 data lines)
      .
            |                | a<MORE> |     3 | $12,500.01 |    61 | male   | d
            |                | c<MORE> |    23 | $12,400.01 |    53 |        | s
            |                | jupiter |    19 | $ 4,100.01 |    47 |        | m

214                                  - Page 1 -
```

This ends the first page of the report (refer to line 151 that set up a two-page display). The second page of the report immediately follows the commentary describing the setup for page 1.

line 145
List the columns currently set for grouping purposes.

line 146
System display -- no current grouping set.

line 148
Set grouping for columns (state, city, and sex), and list the columns currently set as candidates for duplicate suppression.

line 149
System display -- no current outline set.

line 151
Set the outline column value to "sex." The outline option is used to suppress duplicate columns. Outlining is done when the value of a column is the same for the current row as it is on the previous row. Outlining is never done when it is the first row of a new page. The example sets outlining for the sex column. The sex column is the most minor column in the group and therefore all columns more major have outlining done also. The second request on the line invokes display (with sort) of pages 1 and 2. First the data has to be sorted so that use of this option can be further described in later examples.

This page intentionally left blank.

The following example is page 2 of the report invoked by the second request on line 151.

04/29/83 Sample Report 10:33

STATE	CITY	NAME	JOB	SALARY	AGE	SEX	FAMILY
ca	fresno	leeland	14	$32,800.01	77	male	d
		m<MORE>	9	$10,200.01	32		s
		mcclung	5	$13,100.01	71		m
		m<MORE>	1	$14,100.01	26		d
		monger	21	$12,600.01	61		s
	los angeles	aboe	4	$10,201.01	41	female	s
.							
. (37 data lines)							
.							
	san diego	abraham	6	$15,000.01	25	female	d
		c<MORE>	2	$13,000.01	44		s
		kang	22	$19,201.01	23		m
		levy	18	$10,800.01	66		d
		m<MORE>	13	$14,800.01	71		s
		mccrary	8	$13,000.01	25		m

- Page 2 -

Sorting is done completely within the report writer. The values must all be retrieved from the subsystem before sorting can be done. When display is invoked without control arguments, the system defaults to a new retrieve on each invocation. The next two examples show how this retrieve can be kept and then recalled.

```
216 ! display_employee: di -sort state city sex -kr -pg 2

220   04/29/83                    Sample Report                        10:34

      STATE |      CITY      |  NAME    |  JOB  |   SALARY    |  AGE  |  SEX    | F
            |                |          |       |             |       |         | A
            |                |          |       |             |       |         | M
            |                |          |       |             |       |         | I
            |                |          |       |             |       |         | L
            |                |          |       |             |       |         | Y

      ca    | fresno         | leeland  |    14 | $32,800.01  |    77 | male    | d
            |                | m<MORE>  |     9 | $10,200.01  |    32 |         | s
            |                | mcclung  |     5 | $13,100.01  |    71 |         | m
            |                | m<MORE>  |     1 | $14,100.01  |    26 |         | d
            |                | monger   |    21 | $12,600.01  |    61 |         | s
234         | los angeles    | aboe     |     4 | $10,201.01  |    41 | female  | s
      .
      .  (37 data lines)
      .
            | san diego      | abraham  |     6 | $15,000.01  |    25 | female  | d
            |                | c<MORE>  |     2 | $13,000.01  |    44 |         | s
            |                | kang     |    22 | $19,201.01  |    23 |         | m
            |                | levy     |    18 | $10,800.01  |    66 |         | d
            |                | m<MORE>  |    13 | $14,800.01  |    71 |         | s
            |                | mccrary  |     8 | $13,000.01  |    25 |         | m

279                                 - Page 2 -
```

line 216
: Sort the state, city, and sex columns; then display page 2. In addition, keep the results of the retrieve.

lines 220–279
: System display.

The sorted data is now retained for future use (see -kr on line 216). Future display requests may now re-call the kept data (i.e., the amount of system time required after execution of line 216 until the report is displayed can be minimized in future displays).

```
281 ! display_employee:  di -kr -or -pg 2
```

The display results (provided by execution of line 281) would be an exact copy of that provided in lines 220–279 above, except that the time required to produce the report is less.

Outlining can also be done on columns which are not a member of the group. For example:

```
283 ! display_employee:  lsfo -out
      -outline                    "sex"

286 ! display_employee:  sfo -out sex family
288 ! display_employee:  di -kr -or -pg 1,2

292   04/29/83                    Sample Report                          10:36

      STATE |      CITY      |  NAME   |  JOB  |   SALARY   |  AGE  |  SEX   | F
            |                |         |       |            |       |        | A
          . |                |         |       |            |       |        | M
            |                |         |       |            |       |        | I
            |                |         |       |            |       |        | L
            |                |         |       |            |       |        | Y

      ak    | juneau         | bambry  |    10 | $11,501.01 |    66 | female | d
            |                | gaskins |     6 | $14,700.01 |    31 |        | s
            |                | justin  |     2 | $12,000.01 |    78 |        | m
      .
      .  (16 data lines)
      .
      az    | phoenix        | abell   |     2 | $13,000.01 |    55 | female | m
            |                | c<MORE> |    22 | $18,300.01 |    38 |        | d
            |                | june    |    18 | $10,900.01 |    73 |        | s
      .
      .  (12 data lines)
      .
            | tucson         | monaco  |    20 | $12,300.01 |    30 | female | d
            |                | nevitte |    15 | $12,300.01 |    77 |        | s
            |                | pauley  |    10 | $11,600.01 |    56 |        | m
338         |                | n<MORE> |     5 | $12,400.01 |    57 | male   |
            |                | ordeman |     1 | $15,200.01 |    21 |        | d
      ca    | fresno         | bane    |    13 | $15,200.01 |    50 | female | m
      .
      .  (6 data lines)
      .
            |                | a<MORE> |     3 | $12,500.01 |    61 | male   | d
            |                | c<MORE> |    23 | $12,400.01 |    53 |        | s
            |                | jupiter |    19 | $ 4,100.01 |    47 |        | m

351                                  - Page 1 -
```

line 283
List the columns currently set as candidates for duplicate suppression.

line 286
Set the outline column value to "sex" and "family." (Refer to additional description regarding outlining in the commentary of line 151 above.)

line 288
Display page 1 and 2 using the data retrieved during the previous invocation (-or), and keep the retrieved data (-kr) from this execution for use in subsequent invocations of the display request.

lines 292-351
System display -- note that the family entry for line 338 is blank indicating duplicate suppression of "m" which would normally have displayed (see line 201 above).

Page 2 of the report is not shown.

The size of a retrieved table can cause a process directory quota overflow when working with large tables. The -temp_dir control argument for the display request allows the user to provide a directory for the retrieved table where enough quota is available. The -temp_dir argument can only be used when requesting a new table.

```
353 ! display_employee:  di -or -kr -td [e wd] -pg 1
354   display_employee (display): Warning: The temp_dir
                >udd>Demo>display_employee won't be used.
```

line 353
: Display page 1 using the data retrieved during the previous invocation (-or), and keep the retrieved data (-kr) from this execution utilizing the temporary directory "wd".

line 354
: System display -- warning message because a new retrieval was not requested (i.e., -old retrieval was used).

Page 1 of the report is not shown. It would be an exact duplicate of that shown in lines 292-351 above.

```
356 ! display_employee:  di -kr -td [e wd] -pg 1 -sort state
              city sex
```

line 356
: Display page 1 using a new retrieval, keep the retrieved data for future use, and utilize "wd" for a temporary directory.

Page 1 of the report is not shown. It would be an exact duplicate of that shown in lines 292-351 above.

To verify that the working directory (wd) was in fact used for the temporary directory, enter:

```
358 ! display_employee: e ls
359   Segments = 224, Length = 353
360
361   rew     0  !BBBJNHFGnQJX1w.temp.0565
                .
                .  (data lines)
                .
369   r w     0  !BBBJNHFGmXFcFB.RW.table
370   r w     1  EXAMPLE-1.fo.display_employee
371   r w     1  sex_lookup.display_employee
                .
                .
                .
```

---

line 358
Escape out of display_employee and list the current contents of the working directory.

lines 359-371
System display -- lines 359-369 outlines the areas used for the temporary directory. Note that line 370 is associated with an earlier example where the contents of a report was saved (refer to "Saving a Report and Resetting Options"), and line 371 identifies the segment which contains the exec_com used to change "m" and "f" to "male" and "female" for the sex column (refer to "Special Editing of a Report").

```
375 ! display_employee:  lsfo -pb
376   -page_break                ""

378 ! display_employee:  sfo -pb state;di -kr -or -pg 1,4
```

---

lines 375-378
is a request to list the current columns that are candidates for new page breaks, and line 376 says there are no current candidates. The following four examples show full-page representations of the results of the requests in line 378 (set page break value to "state" and display pages 1 through 4).

STATE	CITY	NAME	JOB	SALARY	AGE	SEX	F A M I L Y
ak	juneau	bambry	10	$11,501.01	66	female	d
		gaskins	6	$14,700.01	31		s
		justin	2	$12,000.01	78		m
		macleod	22	$18,500.01	43		d
		manuel	18	$10,000.01	33		s
		m<MORE>	13	$14,900.01	67		m
		m<MORE>	8	$13,000.01	77		d
		nesline	4	$10,100.01	27		s
		ord	24	$ 9,200.01	34		m
		abel	1	$14,555.01	36	male	s
		cooke	21	$12,100.01	34		m
		jones	16	$13,000.01	21		d
		ledger	11	$21,900.01	27		s
		maclure	7	$14,700.01	53		m
		m<MORE>	3	$12,100.01	71		d
		mead	23	$12,700.01	29		s
		molloy	19	$ 4,300.01	22		m
		nevling	14	$32,500.01	63		d
		paul	9	$10,300.01	73		s

.
. (10 blank lines)
.

04/29/83 Sample Report 10:40

STATE	CITY	NAME	JOB	SALARY	AGE	SEX	FAMILY
az	phoenix	abell	2	$13,000.01	55	female	m
		c<MORE>	22	$18,300.01	38		d
		june	18	$10,900.01	73		s
		lednar	13	$15,000.01	71		m
		m<MORE>	8	$12,600.01	37		d
		m<MORE>	4	$10,800.01	68		s
		meadow	24	$ 9,800.01	52		m
		bander	11	$21,100.01	70	male	s
		geist	7	$14,600.01	21		m
		kane	3	$12,300.01	58		d
		maclin	23	$12,500.01	79		s
		manzo	19	$ 4,200.01	74		m
		mccoy	14	$31,300.01	67		d
		meagher	9	$10,500.01	52		s
		dupuis	12	$12,000.00	28		y
	tucson	monaco	20	$12,300.01	30	female	d
		nevitte	15	$12,300.01	77		s
		pauley	10	$11,600.01	56		m
		n<MORE>	5	$12,400.01	57	male	
		ordeman	1	$15,200.01	21		d

.

. (10 blank lines)

.

STATE	CITY	NAME	JOB	SALARY	AGE	SEX	FAMILY
ca	fresno	bane	13	$15,200.01	50	female	m
		george	8	$12,100.01	44		d
		kang	4	$10,000.01	76		s
		maclure	24	$ 9,700.01	47		m
		marcey	20	$12,600.01	71		d
		mccrary	15	$12,500.01	53		s
		meakin	10	$11,600.01	51		m
		a<MORE>	3	$12,500.01	61	male	d
		c<MORE>	23	$12,400.01	53		s
		jupiter	19	$ 4,100.01	47		m
		leeland	14	$32,800.01	77		d
		m<MORE>	9	$10,200.01	32		s
		mcclung	5	$13,100.01	71		m
		m<MORE>	1	$14,100.01	26		d
		monger	21	$12,600.01	61		s
	los angeles	aboe	4	$10,201.01	41	female	
		cowes	24	$ 9,500.01	58		m
		justin	20	$12,900.01	34		d
		leestma	15	$12,300.01	69		s
		m<MORE>	10	$11,400.01	52		m
		m<MORE>	6	$15,000.01	26		d

. (10 data lines)

		mealey	11	$21,600.01	36		d
		nevitte	7	$14,900.01	39		s
		orf	3	$12,400.01	70		m
	sacramento	barrett	15	$12,800.01	65	female	s
		gill	10	$11,800.01	47		m
		keene	6	$14,100.01	54		d
		m<MORE>	2	$12,200.01	54		s
		marcy	22	$19,700.01	45		m
		m<MORE>	18	$10,900.01	62		d
		means	13	$14,300.01	46		s
		newcomb	8	$12,300.01	36		m
		orlaens	4	$10,300.01	41		d
		a<MORE>	5	$12,900.01	61	male	m
		c<MORE>	1	$14,300.01	50		d
		kane	21	$12,400.01	24		s
		leonard	16	$12,900.01	25		m
		macnabb	11	$21,500.01	68		d
		mccoy	7	$14,000.01	77		s

STATE	CITY	NAME	JOB	SALARY	AGE	SEX	FAMILY
ca	sacramento	meakin	3	$12,900.01	71	male	m
		monson	23	$13,000.01	40		d
		newman	19	$ 4,200.01	68		s
		payne	14	$30,400.01	30		m
	san diego	abraham	6	$15,000.01	25	female	d
		c<MORE>	2	$13,000.01	44		s
		kang	22	$19,201.01	23		m
		levy	18	$10,800.01	66		d
		m<MORE>	13	$14,800.01	71		s
		mccrary	8	$13,000.01	25		m
		mealey	4	$10,700.01	71		d
		montano	24	$ 9,300.01	22		s
		newton	20	$13,100.01	24		m
		peacock	15	$12,500.01	76		d
		b<MORE>	16	$12,310.01	63	male	m
		keener	7	$14,000.01	62		s
		m<MORE>	3	$12,400.01	63		m
		m<MORE>	23	$12,600.01	38		d
		m<MORE>	19	$ 4,000.01	22		s
		mecham	14	$30,400.01	23		m
		newhall	9	$10,300.01	21		d
		o<MORE>	5	$12,900.01	27		s
	san francisco	baur	18	$10,100.01	79	female	d

. (8 data lines)

		abrahms	7	$14,300.01	35	male	s
		c<MORE>	3	$12,501.01	76		m
		katz	23	$12,500.01	58		d
		libin	19	$ 4,000.01	29		s
		macnair	14	$31,300.01	70		m
		mccory	9	$10,500.01	52		d
		means	5	$12,900.01	60		s
		monte	1	$15,300.01	31		m
		nguyen	21	$12,700.01	53		d
		parce	16	$12,900.01	68		s
	santa cruz	nevling	6	$14,400.01	37	female	d
		orend	2	$12,900.01	72		s
		newcomb	16	$12,400.01	72	male	m
		paulson	11	$21,100.01	49		d

This page intentionally left blank.

Now we will experiment with column subtotals and totals. A subtotal specification is given in the form of one or more "triplets." A triplet is given as the column to be subtotaled, followed by the column whose value change should generate the subtotal, and optionally followed by "reset" or "running" to indicate what type of subtotal is desired. Reset is the default. In the following example, lines 1-8 are intentionally left blank.

```
 9 ! display_employee:  sfo -rhv "" -rfv ""
11 ! display_employee:  lsfo -stt
12   -subtotal                  ""

14 ! display_employee:  sfo -stt salary,state,reset
```

line 9
: Set row header and row footer values to "default."

lines 11-14
: List current value for subtotal, and set up new value.

The subtotal inherits its width, editing request, etc. from the parent column. The width of the salary column must be increased or the subtotal is folded, and a larger picture is needed to edit it through. The age and job columns are left at their present width so the filling of numbers can be seen later when the numbers become large enough.

```
16 ! display_employee:  lsfo -wid salary
17   -width salary              "10"

19 ! display_employee:  sfo -wid salary 14
21 ! display_employee:  lsfo -ed salary
22   -editing salary            "[e pic $zz,zz9v.99 [clv salary]]"

24 ! display_employee:  sfo -ed salary "[e pic $zz,zzz,zz9v.99 [clv salary]]"
26 ! display_employee:  lsfo -al salary
27   -alignment salary          "decimal 8"

29 ! display_employee:  sfo -al salary decimal 12
31 ! display_employee:  di -nr -kr -sort state city sex -pg 1,4

35   04/29/83                          Sample Report                          10:42

     STATE |      CITY       |  NAME   |  JOB  |      SALARY       |  AGE  |  SEX   | F
           |                 |         |       |                   |       |        | A
           |                 |         |       |                   |       |        | M
           |                 |         |       |                   |       |        | I
           |                 |         |       |                   |       |        | L
           |                 |         |       |                   |       |        | Y

     ak    | juneau          | bambry  |    10 | $     11,501.01  |    66 | female | d
           |                 | gaskins |     6 | $     14,700.01  |    31 |        | s
           |                 | justin  |     2 | $     12,000.01  |    78 |        | m
     .
     .  (15 data lines)
     .
           |                 | paul    |     9 | $     10,300.01  |    73 |        | s
           |                 |         |       | ----------------- |       |        |
64   ak    |                 |         |       | $    262,056.19  |       |        |
     .
     .  (30 blank lines)
     .
95                                    - Page 1 -
```

lines 16-29
List current value for width, editing, and alignment of the salary column, and set up new values.

line 31
Display pages 1 through 4 of the report, starting with a new retrieval, sorting the report as indicated to get back into the full format, and keep the retrieval for re-use.

lines 35-281
System display -- note the inclusion of subtotals in the salary column (total by state -- see lines 64, 127, and 276). The remaining three pages of the report follow.

```
97   04/29/83                          Sample Report                                  10:42

     STATE |      CITY       |  NAME    |  JOB  |      SALARY      |  AGE  |  SEX    | F
           |                 |          |       |                  |       |         | A
           |                 |          |       |                  |       |         | M
           |                 |          |       |                  |       |         | I
           |                 |          |       |                  |       |         | L
           |                 |          |       |                  |       |         | Y

     az    | phoenix         | abell    |     2 | $    13,000.01 |    55 | female  | m
           |                 | c<MORE>  |    22 | $    18,300.01 |    38 |         | d
     .
     .  (13 data lines)
     .
           | tucson          | monaco   |    20 | $    12,300.01 |    30 | female  | d
           |                 | nevitte  |    15 | $    12,300.01 |    77 |         | s
           |                 | pauley   |    10 | $    11,600.01 |    56 |         | m
           |                 | n<MORE>  |     5 | $    12,400.01 |    57 | male    |
           |                 | ordeman  |     1 | $    15,200.01 |    21 |         | d
           |                 |          |       | -------------- |       |         |
127  az    |                 |          |       | $   272,700.19 |       |         |
     .
     .  (31 blank lines)
     .
157                                    - Page 2 -

159  04/29/83                          Sample Report                                  10:42

     STATE |      CITY       |  NAME    |  JOB  |      SALARY      |  AGE  |  SEX    | F
           |                 |          |       |                  |       |         | A
           |                 |          |       |                  |       |         | M
           |                 |          |       |                  |       |         | I
           |                 |          |       |                  |       |         | L
           |                 |          |       |                  |       |         | Y

     ca    | fresno          | bane     |    13 | $    15,200.01 |    50 | female  | m
     .
     .  (14 data lines)
     .
           | los angeles     | aboe     |     4 | $    10,201.01 |    41 | female  |
           |                 | cowes    |    24 | $     9,500.01 |    58 |         | m
     .
     .  (17 data lines)
     .
           | sacramento      | barrett  |    15 | $    12,800.01 |    65 | female  | s
           |                 | gill     |    10 | $    11,800.01 |    47 |         | m
     .
     .  (12 data lines)
     .
           |                 | mccoy    |     7 | $    14,000.01 |    77 |         | s

219                                    - Page 3 -
```

STATE	CITY	NAME	JOB	SALARY	AGE	SEX	F A M I L Y
ca	sacramento	meakin	3	$ 12,900.01	71	male	m
		monson	23	$ 13,000.01	40		d
		newman	19	$ 4,200.01	68		s
	.	payne	14	$ 30,400.01	30		m
	san diego	abraham	6	$ 15,000.01	25	female	d
		c<MORE>	2	$ 13,000.01	44		s

. (16 data lines)

	san francisco	baur	18	$ 10,100.01	79	female	d
		gnandt	13	$ 14,700.01	60		s

. (17 data lines)

	santa cruz	nevling	6	$ 14,400.01	37	female	d
		orend	2	$ 12,900.01	72		s
		newcomb	16	$ 12,400.01	72	male	m
		paulson	11	$ 21,100.01	49		d
				---			
276 ca				$ 1,281,323.94			

. (4 blank lines)

The following example shows how to get subtotals for multiple columns in addition to more than one subtotal per column.

```
 1 ! display_employee:  sfo -stt -prompt
 2   Enter -subtotal.
 3 ! age,sex salary,sex job,sex age,city salary,city job,city age,state
 4 ! salary,state job,state
 5 ! .

 7 ! display_employee:  di -or -kr -pg 1,3

11   04/29/83                        Sample Report                          10:43

     STATE |      CITY       |  NAME   |  JOB  |      SALARY      |  AGE  |  SEX   | F
           |                 |         |       |                  |       |        | A
           |                 |         |       |                  |       |        | M
           |                 |         |       |                  |       |        | I
           |                 |         |       |                  |       |        | L
           |                 |         |       |                  |       |        | Y

     ak    | juneau          | bambry  |    10 | $    11,501.01  |    66 | female | d
           |                 | gaskins |     6 | $    14,700.01  |    31 |        | s
           |                 | justin  |     2 | $    12,000.01  |    78 |        | m
           |                 | macleod |    22 | $    18,500.01  |    43 |        | d
           |                 | manuel  |    18 | $    10,000.01  |    33 |        | s
           |                 | m<MORE> |    13 | $    14,900.01  |    67 |        | m
           |                 | m<MORE> |     8 | $    13,000.01  |    77 |        | d
           |                 | nesline |     4 | $    10,100.01  |    27 |        | s
           |                 | ord     |    24 | $     9,200.01  |    34 |        | m
           |                 |         | -----  | -------------- | -----  |        |
30         |                 |         |   107 | $   113,901.09  |   456 | female |

           |                 | abel    |     1 | $    14,555.01  |    36 | male   | s
     .
     .  (5 data lines)
     .
           |                 | mead    |    23 | $    12,700.01  |    29 |        | s
           |                 | molloy  |    19 | $     4,300.01  |    22 |        | m
           |                 | nevling |    14 | $    32,500.01  |    63 |        | d
           |                 | paul    |     9 | $    10,300.01  |    73 |        | s
           |                 |         | -----  | -------------- | -----  |        |
43   ak    | juneau          |         |   124 | $   148,155.10  |   429 | male   |

           |                 |         | -----  | -------------- | -----  |        |
46   ak    | juneau          |         |   231 | $   262,056.19  |   885 |        |

           |                 |         | -----  | -------------- | -----  |        |
49   ak    |                 |         |   231 | $   262,056.19  |   885 |        |
     .
     .  (22 blank lines)
     .

71                                  - Page 1 -
```

line 1
Request to set columns for subtotaling, with prompt.

line 2
System display --prompt.

lines 3-5
Set column values to be subtotaled, and terminate the prompt.

line 7
Display pages 1 through 3.

lines 11-195
System display -- the display of pages 2-3 follow.

STATE	CITY	NAME	JOB	SALARY	AGE	SEX	FAMILY
az	phoenix	abell	2	$ 13,000.01	55	female	m
		c<MORE>	22	$ 18,300.01	38		d
		june	18	$ 10,900.01	73		s
		lednar	13	$ 15,000.01	71		m
		m<MORE>	8	$ 12,600.01	37		d
		m<MORE>	4	$ 10,800.01	68		s
		meadow	24	$ 9,800.01	52		m
			---	---	---		
			91	$ 90,400.07	394	female	
		bander	11	$ 21,100.01	70	male	s
		geist	7	$ 14,600.01	21		m
		kane	3	$ 12,300.01	58		d
		maclin	23	$ 12,500.01	79		s
		manzo	19	$ 4,200.01	74		m
		mccoy	14	$ 31,300.01	67		d
		meagher	9	$ 10,500.01	52		s
		dupuis	12	$ 12,000.00	28		y
			---	---	---		
	phoenix		98	$ 118,500.07	449	male	
			---	---	---		
	phoenix		189	$ 208,900.14	843		
	tucson	monaco	20	$ 12,300.01	30	female	d
		nevitte	15	$ 12,300.01	77		s
		pauley	10	$ 11,600.01	56		m
			---	---	---		
			45	$ 36,200.03	163	female	
		n<MORE>	5	$ 12,400.01	57	male	
		ordeman	1	$ 15,200.01	21		d
			---	---	---		
az	tucson		6	$ 27, 60.02	78	male	
			---	---	---		
az	tucson		51	$ 63,800.05	241		
			---	---	---		
az			240	$ 272,700.19	1084		

. (10 blank lines)

STATE	CITY	NAME	JOB	SALARY	AGE	SEX	FAMILY
ca	fresno	bane	13	$ 15,200.01	50	female	m
		george	8	$ 12,100.01	44		d
		kang	4	$ 10,000.01	76		s
		maclure	24	$ 9,700.01	47		m
		marcey	20	$ 12,600.01	71		d
		mccrary	15	$ 12,500.01	53		s
		meakin	10	$ 11,600.01	51		m
			94	$ 83,700.07	392	female	
		a<MORE>	3	$ 12,500.01	61	male	d
		c<MORE>	23	$ 12,400.01	53		s
		jupiter	19	$ 4,100.01	47		m
		leeland	14	$ 32,800.01	77		d
		m<MORE>	9	$ 10,200.01	32		s
		mcclung	5	$ 13,100.01	71		m
		m<MORE>	1	$ 14,100.01	26		d
		monger	21	$ 12,600.01	61		s
	fresno		95	$ 111,800.08	428	male	
	fresno		189	$ 195,500.15	820		
	los angeles	aboe	4	$ 10,201.01	41	female	
		cowes	24	$ 9,500.01	58		m
		justin	20	$ 12,900.01	34		d
		leestma	15	$ 12,300.01	69		s
		m<MORE>	10	$ 11,400.01	52		m
		m<MORE>	6	$ 15,000.01	26		d
		meagher	2	$ 12,600.01	67		s
		monroe	22	$ 18,900.01	42		m
		newhall	18	$ 10,000.01	30		d
		pavlov	13	$ 14,000.01	24		s
			134	$ 126,801.10	443	female	
		barker	14	$ 32,800.01	78	male	d

. (7 data lines)

This page intentionally left blank.

To see how the totals feature works, the last page of the report must be examined. The example eliminates page breaks to cut down on the number of pages generated.

```
197 ! display_employee:  sfo -tt age salary job
199 ! display_employee:  sfo -pb ""
```

Just as retrieved data can be re-used, so can formatted reports. The last few pages will be examined, but display will be asked to keep the formatted report. It will use the previously established temp_dir to place the copy of the formatted report.

```
201 ! display_employee:  di -kr -or -krp -pg 33,$
```

04/29/83 Sample Report 10:52

STATE	CITY	NAME	JOB	SALARY	AGE	SEX	FAMILY
vt	rutland	parnell	1	$ 14,400.01	59	male	m
			-----	----------	-----		
vt	rutland		92	$ 133,200.09	466	male	
			-----	----------	-----		
vt	rutland		218	$ 246,511.18	992		
			-----	----------	-----		
vt			431	$ 478,511.36	1866		
wa	seattle	aziz	6	$ 14,100.01	75	female	
		freitag	2	$ 13,000.01	66		d
		johnson	22	$ 18,900.01	25		s
		maclean	18	$ 10,000.01	74		m
		m<MORE>	13	$ 14,000.01	67		d
		m<MORE>	8	$ 12,600.01	72		s
		m<MORE>	4	$ 10,200.01	67		m
		neff	24	$ 9,100.01	65		d
		o'neil	20	$ 12,100.01	49		s
			-----	----------	-----		
			117	$ 114,000.09	560	female	
		collier	16	$ 13,000.01	62	male	
		janick	11	$ 20,200.01	50		m
		latter	7	$ 15,300.01	55		d
		m<MORE>	3	$ 12,400.01	66		s
		m<MORE>	23	$ 12,500.01	43		m
		mcrorie	19	$ 4,000.01	40		d
		mock	14	$ 30,100.01	22		s
		neill	9	$ 10,800.01	47		m
		patel	5	$ 12,000.01	24		d
			-----	----------	-----		
	seattle		107	$ 130,300.09	409	male	
			-----	----------	-----		
	seattle		224	$ 244,300.18	969		

.
. (13 data lines)
.

STATE	CITY	NAME	JOB	SALARY	AGE	SEX	FAMILY
wa	walla walla	bahn	7	$ 14,900.01	55	male	d
		freuh	3	$ 12,900.01	76		s
		jones	23	$ 12,800.01	36		m
		macleod	19	$ 4,200.01	60		d
		manion	14	$ 32,400.01	68		s
		m<MORE>	9	$ 10,700.01	68		m
		mittal	5	$ 12,000.01	43		d
		negri	1	$ 15,200.01	68		s
		oong	21	$ 12,000.01	70		m
			---	---	---		
wa	walla walla		102	$ 127,100.09	544	male	
			---	---	---		
wa	walla walla		220	$ 236,301.18	929		
			---	---	---		
wa			444	$ 480,601.36	1898		
wi	green bay	bailey	8	$ 12,410.01	25	female	s
		fyock	4	$ 10,801.01	47		m
		june	24	$ 9,210.01	47		d
		m<MORE>	20	$ 12,800.01	35		s
		mann	15	$ 13,000.01	52		m
		m<MORE>	10	$ 12,000.01	51		d
		mock	6	$ 14,800.01	78		s
		neill	2	$ 12,200.01	30		m
		onofiro	22	$ 18,800.01	77		d
			---	---	---		
			111	$ 116,021.09	442	female	

. (11 data lines)

		---	---	---	---	
green bay		103	$ 132,200.09	473	male	
		---	---	---	---	
green bay		214	$ 248,221.18	915		
racine	c<MORE>	20	$ 12,500.01	20	female	s
	johnson	15	$ 12,410.01	61		m
	ledford	10	$ 11,200.01	63		d
	maclin	6	$ 14,200.01	25		s

STATE	CITY	NAME	JOB	SALARY	AGE	SEX	FAMILY
wi	racine	m<MORE>	2	$ 12,600.01	35	female	m
		m<MORE>	22	$ 18,700.01	38		d
		moldt	18	$ 10,200.01	67		s
		n<MORE>	13	$ 14,400.01	71		m
		patton	8	$ 12,100.01	27		d
			---	---	---		
			114	$ 118,310.09	407	female	
		baker	9	$ 12,000.10	43	male	m
		gardner	5	$ 12,300.01	29		d
		jupiter	1	$ 14,301.01	41		s
		m<MORE>	21	$ 12,600.01	32		m
		mansour	16	$ 13,100.01	46		d
		mcclung	11	$ 21,100.01	39		s
		modlin	7	$ 14,300.01	60		m
		nelson	3	$ 12,400.01	70		d
		o<MORE>	23	$ 12,800.01	62		s
			---	---	---		
wi	racine		96	$ 124,901.18	422	male	
			---	---	---		
wi	racine		210	$ 243,211.27	829		
			---	---	---		
wi			424	$ 491,432.45	1744		
			=====	==========	=====		
			12279	$13,766,244.12	48952		

. (10 blank lines)

Now that the report appears correct, it can be written (saved) to a file. -old_report will be specified so that display uses the previously formatted report.

```
203 ! display_employee:  di -orp -of SAMPLE_REPORT -kr
```

The complete report (SAMPLE_REPORT) now resides in the user's working directory and can be dprinted at will. The -keep_retrieval control argument was specified in order to continue this session, but could have been eliminated if the user was terminating the session after saving this report.

Now we will experiment with generation of a report utilizing the group footer/header and left/right trim operations.

```
 1 ! display_employee:  sfo -rs
 3 ! display_employee:  sfo -pw 60 -tl off -pb state
 5 ! display_employee:  sfo -ex 1 2 3 4 5 6 7 8 -gr state city
 7 ! display_employee:  sfo -gft city -ght city

 9 ! display_employee:  sfo -gfv -prompt -ghv -prompt
10   Enter -group_footer_value.
11 ! !!!!
12 ! .
13   Enter -group_header_value.
14 ! !City: [clv city]!!!
15 ! !!!!
16 ! .

18 ! display_employee:  sfo -phv -prompt -pfv -prompt
19   Enter -page_header_value.
20 ! !State: [clv state]!!!
21 ! !!!!
22 ! .
23   Enter -page_footer_value.
24 ! !!!!
25 ! !!- Page [dib page_number] -!!
26 ! .

28 ! display_employee:  sfo -rhv -prompt
29   Enter -row_header_value.
30 ! Employee [e rtrim [clv name]] is [e ltrim [clv age]] years old and
        earns [e pic $z9,999v.99 [clv salary]]!!!
31 ! .

33 ! display_employee: di -or -kr -sort state city salary -pg 1,3

37   State: ak

39   City: juneau

40   Employee molloy is 22 years old and earns $ 4,300.01
     Employee ord is 34 years old and earns $ 9,200.01
     Employee manuel is 33 years old and earns $10,000.01
     Employee nesline is 27 years old and earns $10,100.01
     Employee paul is 73 years old and earns $10,300.01
     .
     .  (12 data lines)
     .
     Employee ledger is 27 years old and earns $21,900.01
     Employee nevling is 63 years old and earns $32,500.01
     .
     . (37 blank lines)
     .
97                          - Page 1 -
```

lines 1-5
Resets all options (i.e., restore the report back to its original format), set page width to 60, turn title line "off," set the page break to "state," exclude all 8 columns of the report, and group the report by "state" and "city."

line 7
Sets the group footer/header trigger to "city."

lines 9-16
Sets the group footer value to a blank line (!!!!) and the group header value to "City:" (left-justified).

lines 18-26
Sets the page header value to "State:" (left-justified), the page footer (2 lines) to contain a blank line (!!!!), and the second footer line to "- Page X -".

lines 28-31
Sets the row header value to read (left-justified and trimmed):

Employee X is X years old and earns $X

line 33
Invokes display, using the sort sequence "state city salary."

lines 37-225
System display -- notice that the top of each page (lines 37, 101, 165) indicate a report by state (ak, az, ca). Additionally, the report is sorted by city, where:

ak - juneau (line 39)
az - phoenix (line 103)
- tucson (line 121)
ca - fresno (line 167)
- los angeles (line 185)
- sacramento (line 207)

and finally employees are listed in ascending salary order.

The remaining two pages of the report follow.

```
101   State:  az

103   City:  phoenix

      Employee manzo is 74 years old and earns $ 4,200.01
      Employee meadow is 52 years old and earns $ 9,800.01
      Employee meagher is 52 years old and earns $10,500.01
      Employee mcclowsky is 68 years old and earns $10,800.01
      Employee june is 73 years old and earns $10,900.01
      Employee dupuis is 28 years old and earns $12,000.00
      Employee kane is 58 years old and earns $12,300.01
      Employee maclin is 79 years old and earns $12,500.01
      Employee macmahon is 37 years old and earns $12,600.01
      Employee abell is 55 years old and earns $13,000.01
      Employee geist is 21 years old and earns $14,600.01
      Employee lednar is 71 years old and earns $15,000.01
      Employee corcoran is 38 years old and earns $18,300.01
      Employee bander is 70 years old and earns $21,100.01
      Employee mccoy is 67 years old and earns $31,300.01

121   City: tucson

      Employee pauley is 56 years old and earns $11,600.01
      Employee monaco is 30 years old and earns $12,300.01
      Employee nevitte is 77 years old and earns $12,300.01
      Employee neubauer is 57 years old and earns $12,400.01
      Employee ordeman is 21 years old and earns $15,200.01
      .
      . (33 blank lines)
      .
161                          - Page 2 -
```

```
165   State: ca

167   City: fresno

      Employee jupiter is 47 years old and earns $ 4,100.01
      Employee maclure is 47 years old and earns $ 9,700.01
      Employee kang is 76 years old and earns $10,000.01
      Employee macmannis is 32 years old and earns $10,200.01
      Employee meakin is 51 years old and earns $11,600.01
      Employee george is 44 years old and earns $12,100.01
      Employee costello is 53 years old and earns $12,400.01
      Employee abernathy is 61 years old and earns $12,500.01
      Employee mccrary is 53 years old and earns $12,500.01
      Employee marcey is 71 years old and earns $12,600.01
      Employee monger is 61 years old and earns $12,600.01
      Employee mcclung is 71 years old and earns $13,100.01
      Employee meadoows is 26 years old and earns $14,100.01
      Employee bane is 50 years old and earns $15,200.01
      Employee leeland is 77 years old and earns $32,800.01

185   City: los angeles

      Employee cowes is 58 years old and earns $ 9,500.01
      Employee newhall is 30 years old and earns $10,000.01
      Employee aboe is 41 years old and earns $10,201.01
      Employee giannoti is 45 years old and earns $10,900.01
      Employee macmillan is 52 years old and earns $11,400.01
      Employee leestma is 69 years old and earns $12,300.01
      Employee katz is 70 years old and earns $12,400.01
      Employee orf is 70 years old and earns $12,400.01
      Employee marcus is 62 years old and earns $12,600.01
      Employee meagher is 67 years old and earns $12,600.01
      Employee mccory is 54 years old and earns $12,700.01
      Employee justin is 34 years old and earns $12,900.01
      Employee pavlov is 24 years old and earns $14,000.01
      Employee macmahon is 57 years old and earns $14,800.01
      Employee nevitte is 39 years old and earns $14,900.01
      Employee mccormick is 26 years old and earns $15,000.01
      Employee monroe is 42 years old and earns $18,900.01
      Employee mealey is 36 years old and earns $21,600.01
      Employee barker is 78 years old and earns $32,800.01

207   City: sacramento

      Employee newman is 68 years old and earns $ 4,200.01
      Employee orlaens is 41 years old and earns $10,300.01
      .
      .  (12 data lines)
      .

225                          - Page 3 -
```

```
229 ! display_employee: q
230   (Multics command level - ready message)
```

This concludes the sample user session.

SECTION 3

SUBROUTINE OVERVIEW AND TUTORIAL

This section provides an overview of the report_writer_ subroutine, including examples organized to provide a tutorial on using the subroutine. The report_writer_ subroutine entrypoints are documented in reference material format in Section 5. A PL/I coded example that uses report_writer_ is in Section 6.

Application subsystems that utilize the report writer do so by interfacing through the report_writer_ subroutine and the subsystem utilities subroutine (ssu_). The report_writer_ and ssu_ provide the necessary subroutine entrypoints for an application subsystem to include the report writer with an minimal amount of coding.

Note: Refer to the Programmer's Reference Manual and the Subroutines Manual for ssu_ reference material and subroutine description.

CREATING AN INVOCATION

The application subsystem creates a report writer invocation through the report_writer_$create_invocation entrypoint. Prior to creating a report writer invocation, the application subsystem must have created an ssu_ invocation through the ssu_$create_invocation or ssu_$standalone_invocation entrypoints. All information specific to the application subsystem is supplied to the ssu_ and report_writer_ entrypoints. After creating the invocation, all of the report writer functions described in Section 1 are available to the application subsystem.

The application subsystem provides a procedure that does the actual retrieval of data from its source data file. This procedure is called the application's "table manager" program, and is called by report_writer_ when rows are needed to format the report. The table manager procedure is described in detail under "Data Table Retrieval" below.

The following diagram (Figure 3-1) shows the relationship between the subsystem, report_writer_, and the table manager.

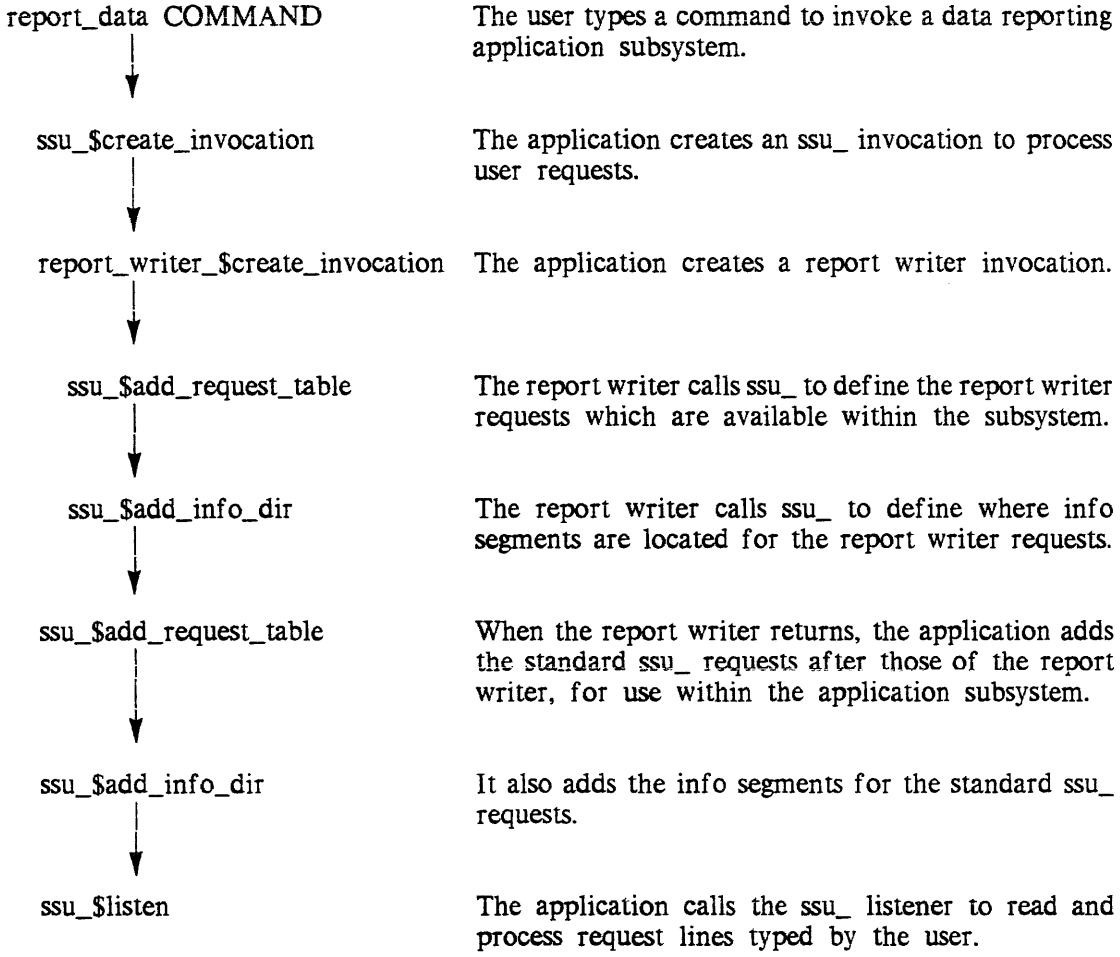


Figure 3-1. Creating an Invocation

The initial step in using report_writer_ is to call the create_invocation entrypoint. This entrypoint takes the name of the table manager procedure from the calling Multics subsystem and an ssu_ info pointer as input parameters, and passes back a pointer to the report_writer_ specific info structure if the invocation can be created. This returned report_writer_ info pointer is saved and passed as the first parameter to every report_writer_ entrypoint to uniquely identify this specific invocation of report_writer_. If the invocation cannot be created, the code and error message parameters supply the reason for the failure to create. The following code fragment shows an example usage of this entrypoint.

```
call report_writer_$create_invocation ("my_table_manager",
     ssu_info_ptr, report_writer_info_ptr, code, message);
if code ^= 0
then call ssu_$abort_line (sci_ptr, code, message);
```

If the table manager procedure name parameter is blank, the create_invocation entrypoint obtains the name of the subsystem via the ssu_$get_subsystem_name subroutine and uses this for the table manager procedure name.

This entrypoint calls ssu_$get_info_ptr to obtain the info pointer for the application subsystem. It is passed as the first parameter to the table manager procedure in the calling subsystem, so that the table manager can access its own info structure and perform operations (refer to "Subsystem Table Manager Procedure" below for additional information).

The report_writer_ also calls ssu_$get_ec_suffix. Any saved report layouts processed by save_format_options, or restore_format_options must have the fo.ec_suffix name, where ec_suffix is the suffix returned by ssu_.

DATA TABLE RETRIEVAL

The application subsystem provides the selection of a table through its own choice of methods. For example, the LINUS subsystem provides the selection of a table through the input_query and translate_query requests.

Before a table has been selected (e.g., by issuing application-provided requests), the requests that deal with format options allow references to general report options, but do not allow references to general or specific column options. When the application subsystem has selected a table, it calls a report_writer_ entrypoint to describe the selected set of columns to the report writer, and report_writer_ generates a default report layout based on these column descriptions. After calling this entrypoint, the requests that deal with format options allow references to general and specific column options.

Although the data selected by the user can come from several different sources (e.g., from several different relations within a MRDS data base), the data that the user eventually sees is considered to be a table made up of one or more columns, that must contain at least one row. This tabular data that the user sees is the same retrieved data table discussed in Sections 1 and 2.

The following diagram (Figure 3-2) shows the relationship between the subsystem, report_writer_, and the table manager.

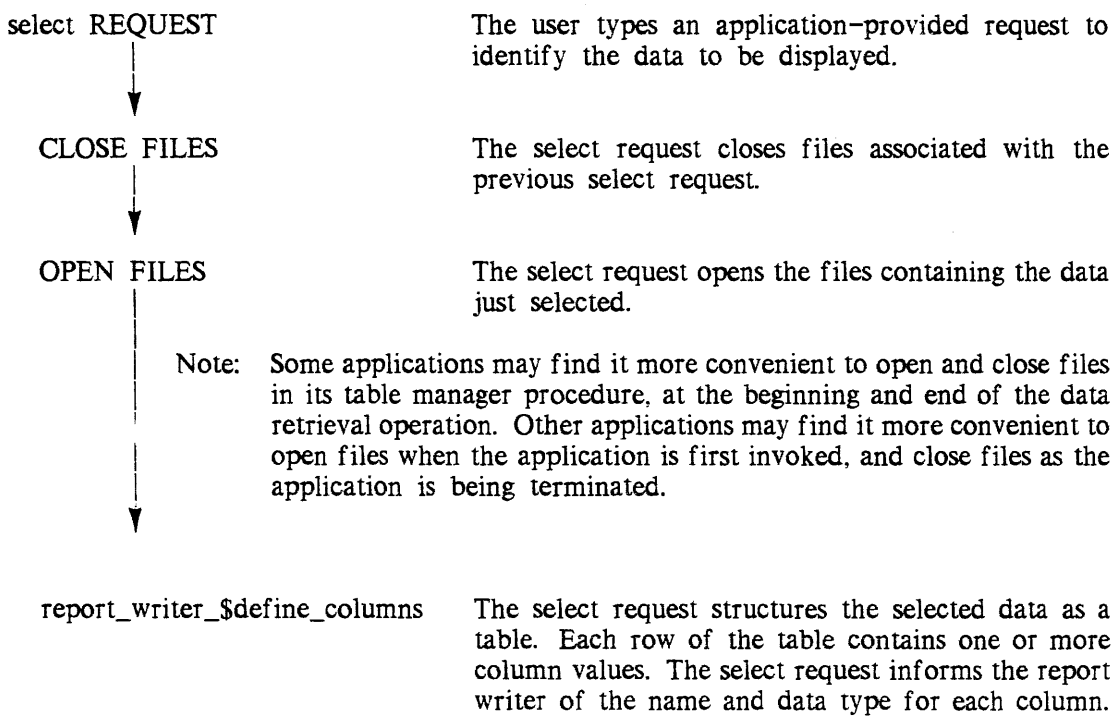


Figure 3-2. Selecting A Table

The application subsystem provides a table manager procedure that prepares for data retrieval, does the actual retrieving of individual rows, and terminates the data retrieval. The report writer calls the subsystem's table manager create_table entrypoint when requested to do a new retrieval, the delete_table entrypoint when requested to delete the retrieval (via display_discard_retrieval), and the get_row entrypoint to retrieve each row of the table during the formatting process. After each get_row operation, report_writer_ takes the data returned from the table manager procedure and places it into an internal location. The sorting of the retrieved data table is done by report_writer_ using its own internal copy. The multi-pass formatting, where several passes over the retrieved data table are necessary, is also done by report_writer_, on its own internal version of the retrieved data (i.e., the get_row entrypoint is only called once for row number 1, regardless of the number of passes over row 1 that are necessary to format the report).

In the case of a subsystem like LINUS, the create_table begins a new data retrieval from MRDS, get_row gets a single row through the dsl_$retrieve subroutine, and delete_table accomplishes whatever cleanup is necessary after the retrieval is completed.

The following diagram (Figure 3-3) shows the relationship between the subsystem, report_writer_, and the table manager.

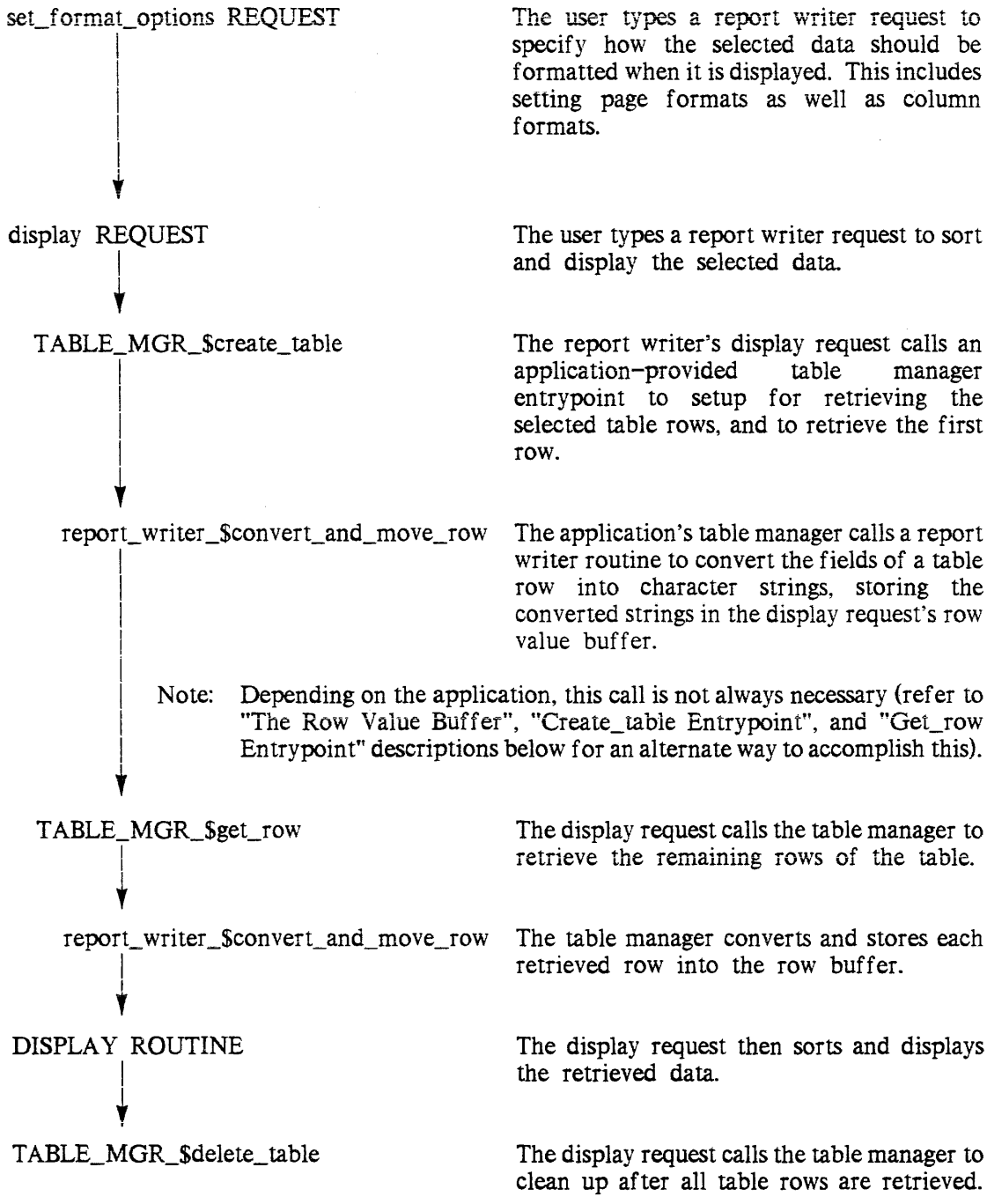


Figure 3-3. Displaying the Data

Data Tables

Reports are created using data which is viewed as a table. The table is made up of one or more columns, and contains one or more rows. No table is defined upon invocation of the report_writer. The selection of any particular table is done by the application subsystem. Until a table is defined, the report_writer_ standard requests do not allow references to any specific or general column options and a report cannot be formatted.

DEFINING A TABLE

After a table is selected by the application subsystem, its select request calls the report_writer_ define_columns entrypoint to inform report_writer_ that a set of columns has been selected. After this call, the standard requests allow references to general and specific column options, and make possible the formatting of reports.

The entrypoint is called with the report_writer_ info pointer as the first parameter and a pointer to the row_info structure as the second parameter. See "Row Information Structure" below for a code fragment initializing this structure and the define_columns entrypoint description in Section 5 for a description of the structure. The code and message parameters reflect any errors that occur during execution.

If the define_columns entrypoint was previously called with a valid set of columns, and some user action results in there being no current set of columns defined, the define_columns entrypoint should be called again with a null pointer to the row_info structure. This causes report_writer_ to delete its information on the previously defined set of columns. Until this is done, report_writer_ considers the old set of columns valid.

The following code fragment illustrates its usage when the application subsystem wishes to define a set of columns.

```
call report_writer_$define_columns (report_writer_info_ptr,
     row_info_ptr, code, message);
if code ^= 0
then call ssu_$abort_line (ssu_info_ptr, code, message);
```

The next code fragment illustrates its usage when there is no defined set of columns.

```
call report_writer_$define_columns (report_writer_info_ptr,
     null, code, message);
if code ^= 0
then call ssu_$abort_line (ssu_info_ptr, code, message);
```

ROW INFORMATION STRUCTURE

The row information structure provides report_writer_ with all of the information about the table needed produce a default report layout, do data conversions, and produce reports. This structure is documented in the define_columns entrypoint in Section 5. The calling program allocates this structure, sets the version number, and fills in the names and descriptors arrays. All other elements of this structure are filled in by the define_columns entrypoint.

The following code fragment shows a portion of a program that uses MRDS, allocates and initializes the row_info structure, and calls report_writer_ to define a set of columns.

```
call dsl_$get_attribute_list (data_base_index, relation_name,
     work_area_ptr, mrds_attribute_list_structure_version,
```

```
     mrds_attribute_list_ptr, code);
if code ^= 0
then call ssu_$abort_line (sci_ptr, code, "Getting data base attributes.");
row_info_init_number_of_columns
     = mrds_attribute_list.num_attrs_in_view;
allocate row_info in (work_area) set (row_info_ptr);
row_info.version = ROW_INFO_VERSION_1;
do loop = 1 to row_info_init_number_of_columns;
     row_info.column (loop).names
          = rtrim (mrds_attribute_list.attribute (loop).model_name);
     row_info.column (loop).descriptors
          = mrds_attribute_list.attribute (loop).user_data_type;
end;
call report_writer_$define_columns (report_writer_info_ptr,
     row_info_ptr, code, message);
if code ^= 0
then call ssu_$abort_line (ssu_info_ptr, code, message);
```

Subsystem Table Manager Procedure

The retrieval of data from the source file is the responsibility of an application provided program (called the "table manager" procedure) which is specified when the invocation is created. The table manager procedure is called by report_writer_ during the formatting process when data to produce the report is needed, and at the beginning and end of the data retrieval process.

When the invocation is created, report_writer_ tries to find the entrypoints create_table, delete_table, get_row, and get_query. The last entrypoint is optional, but the first three entrypoints must be present in the table manager procedure or the invocation is not created. When the mandatory entrypoints are called by report_writer_, they are passed two parameters. The first parameter is the Multics subsystem's info structure pointer which is used by the table manager procedure to reference data items in its info structure necessary to perform its function. The second parameter is an error code which is used to report any errors back to report_writer_. The optional entrypoint is passed these same parameters as the first and last parameters, with the same usage. The second and third parameters are filled in by the table manager procedure if execution completes successfully. The second parameter points to the data selection query used to select the current table, and the third parameter is the query's length.

THE ROW VALUE BUFFER

The row value buffer is the location where the table manager procedure places the row value after it is extracted from the source data file. The maximum row value length supported by report_writer_ is the number of characters that fit in one Multics segment. When the report_writer_$define_columns entrypoint is called with a row_info structure, elements within the structure that deal with the row value buffer are filled in by report_writer_. The row_info.value_ptr is the pointer to the row value buffer, and its length can be found in row_info.value_length. The row_info.column array contains the index into the buffer where each column value is expected to be found by report_writer_, and the column length. Each column within the buffer is treated as a non-varying, unaligned character string.

The declarations in the include file rw_row_info.incl.pl1 provide two methods to access column values. The include file contains the following declaration for the entire row value:

```
declare row_value char (row_info.value_length)
        based (row_info.value_ptr);
```

The following code fragment uses this declaration and the PL/I substring operation to access the column values within the row.

```
do loop = 1 to row_info.number_of_columns;
     call ioa_ ("^a", substr (row_value,
          row_info.column (loop).indexes,
          row_info.column (loop).lengths));
end;
```

The include file also contains the declarations:

```
declare row_value_as_an_array (row_info.value_length)
        char (1) based (row_info.value_ptr);

declare column_value
        char  (row_info.column.lengths
                (row_info.current_column_number))
        based (addr (row_value_as_an_array
                (row_info.column.indexes
                  (row_info.current_column_number))));
```

The next code fragment uses these declarations to access the column values.

```
do row_info.current_column_number = 1 to
     row_info.number_of_columns;
     call ioa_ ("^a", column_value);
end;
```

CREATE_TABLE ENTRYPOINT

The create_table entrypoint of the application's table manager is called by report_writer_ when a new set of retrievals are requested. The table manager procedure may open files, perform initialization, etc. when this entrypoint is called. It also retrieves the first row from its source data file, and moves the row value into the row value buffer. If the source data file contains only character data type columns, the values can be moved directly into the row value buffer. If the source data file contains other data type columns, the report_writer_ entrypoint convert_and_move_row can be called to move the row value into the buffer with data conversions. (Refer to "Data Conversions" below for more information on this entrypoint and to the "get_row Entrypoint" for additional information on moving a row value into the row value buffer without data conversions.)

If the create_table entrypoint finds the source file empty when retrieving the first row, it sets the code to rw_error_$no_data. This is treated as an error by report_writer_, and the setup and creation of the report is terminated. The report_writer_ calls ssu_$abort_line with rw_error_$no_data, and the user of the application subsystem is provided with the following message:

```
subsystem (display): No data was found that satisfied the search.
```

GET_ROW ENTRYPOINT

The get_row entrypoint is called by report_writer_ to retrieve a single row from the source file and move it into the row value buffer. It is called to retrieve each row except the first row (i.e., the first row is returned by the create_table entrypoint). If an end-of-file is discovered by the table manager procedure when the retrieval of the row is attempted, the code should be set to error_table_$end_of_info. This indicates a normal end-of-file and report_writer_ does not call the get_row entrypoint again until a new set of retrievals is requested, via the create_table entrypoint.

The following code fragment shows an example of a structured file containing fixed position, unaligned non-varying character data type columns. The record is read directly into the row value buffer because it is identical to what report_writer_ is expecting.

```
call iox_$read_record (iocb_ptr, row_info.value_ptr,
     row_info.value_length, (0), code_parameter);
return;
```

The next code fragment shows an example of a structured file containing fixed position character data type columns, which are a mixture of varying, non-varying, aligned, and unaligned. The record is read into the structure, and is moved into the row value buffer to end up with unaligned, non-varying column values placed in their correct positions within the row value buffer.

```
declare 1 employee aligned,
          2 number char(8) varying,
          2 street char(64) varying,
          2 city char(32) varying,
          2 state char(2) unaligned,
          2 zip_code char(5) unaligned;
call iox_$read_record (iocb_ptr, addr (employee),
     currentsize (employee) * 4, (0), code_parameter);
if code_parameter ^= 0
then return;
row_info.current_column_number = 1;
column_value = employee.number;
row_info.current_column_number = 2;
column_value = employee.street;
row_info.current_column_number = 3;
column_value = employee.city;
row_info.current_column_number = 4;
column_value = employee.state;
row_info.current_column_number = 5;
column_value = employee.zip_code;
```

DELETE_TABLE ENTRYPOINT

The delete_table entrypoint is called by report_writer_ when the processing of the report is completed. The table manager procedure may close open files, free allocated variables, etc. when this entrypoint is called.

GET_QUERY ENTRYPOINT

The get_query entrypoint is called by report_writer_ when the report_writer_ save_format_options request is invoked with the -query control argument. This specifies to save_format_options that the data selection requests should be saved along with the report layout.

This entrypoint is optional, and if not found in the table manager procedure when the report_writer_ invocation is created, the save_format_options request calls ssu_$abort_line with the code error_table_$unsupported_operation.

The data selection requests are dependent on the application subsystem, and should contain whatever subsystem requests are necessary to reselect the current table. The requests are saved in a subsystem exec_com, along with other requests that set the current report format. A subsequent restore_format_options request invokes the exec_com and executes these requests to reselect the table and restore the current report format. An example of a query specific to the LINUS subsystem is:

```
input_query -force -brief -terminal_input
select * from employee
.
translate_query
```

If any ampersands are found within the query that would be interpreted by exec_com when the report layout is restored, they are protected by save_format_options with double ampersands before being placed in the saved report layout file. In the case of a subsystem like LINUS, a portion of a select statement that looked like:

```
& name = "Smith"
```

would be saved as:

```
&& name = "Smith"
```

The save_format_options request also provides an exec_com &version statement and &attach input lines statement, placed prior to the query and format option request lines.

The entrypoint is called with the calling subsystem info structure pointer as the first parameter. The second parameter is a pointer to a segment where the table manager procedure places the query. The third parameter is for the length of the query in characters, and is set by the table manager procedure. The fourth parameter is a standard error code, and is set to zero if the get_query entrypoint succeeds. If the get_query entrypoint fails, it sets this parameter to a standard error code to indicate the reason for the failure.

Data Conversions

The report_writer_ convert_and_move_row entrypoint can be called by the table manager procedure to have values converted from the original source file data types, to non-varying, unaligned character strings. The converted values are moved into the row value buffer as part of the conversion process.

The following code fragment shows how a record from a structured file can be placed in the row_value buffer when the record contains a mixture of data types. The record is read into a structure, an array of pointers is set to point to the structure elements that make up the column values in column order, and the convert_and_move_row entrypoint is called to perform data conversions and place the converted column values into their correct positions within the row

value buffer. In the example, the column order is, employee.number, employee.assignment, employee.salary, and employee.age.

```
declare 1 employee aligned,
          2 number char(8) varying,
          2 salary fixed dec(7,2) unaligned,
          2 age fixed dec(2) unaligned,
          2 assignment fixed bin;
declare value_ptrs(4) ptr;
call iox_$read_record (iocb_ptr, addr (employee),
     currentsize (employee) * 4, (0), code_parameter);
if code_parameter ^= 0
then return;
value_ptrs (1) = addr (employee.number);
value_ptrs (2) = addr (employee.assignment);
value_ptrs (3) = addr (employee.salary);
value_ptrs (4) = addr (employee.age);
call report_writer_$convert_and_move_row
     (report_writer_info_ptr, value_ptrs);
```

Note that in some cases, the table manager might use only a few of the structure elements as table columns, or might read records from two or more data files and merge them into a single table row, with columns coming from several data files.

REPORT PREPARATION

A default report layout is provided for each selected set of columns as described in Sections 1 and 2. The listing of formatting options is done through the list_format_options request described in Sections 1, 2, and 4. If the appliction subsystem provides a request level interface, the user can type the list_format_options request. If the application subsystem does not provide a request level interface, it must utilize the ssu_$evaluate_active_string entrypoint to obtain the value of a format option. The restore_format_options, save_format_options, and set_format_options requests can be invoked in the application when a user types these requests, or the application can use ssu_$execute_string or ssu_$execute_line to invoke these requests on the user's behalf.

REPORT FORMATTING

The display, display_builtins, and column_value requests provide for the formatting of a report. The display request is invoked in the user typing display, or the application subsystem calling the ssu_$execute_line or ssu_$execute_string to invoke these requests on the user's behalf. The display_builtins and column_value requests are normally accessed by including references to them in the values of format options which are set through the set_format_options request, but can also be accessed during report creation by the application subsystem through the ssu_$evaluate_active_string entrypoint.

DESTROYING AN INVOCATION

The application subsystem destroys a report writer invocation with the report_writer_$destroy_invocation entrypoint.

The following diagram (Figure 3-4) shows the relationship between the subsystem, report_writer_, and the table manager.

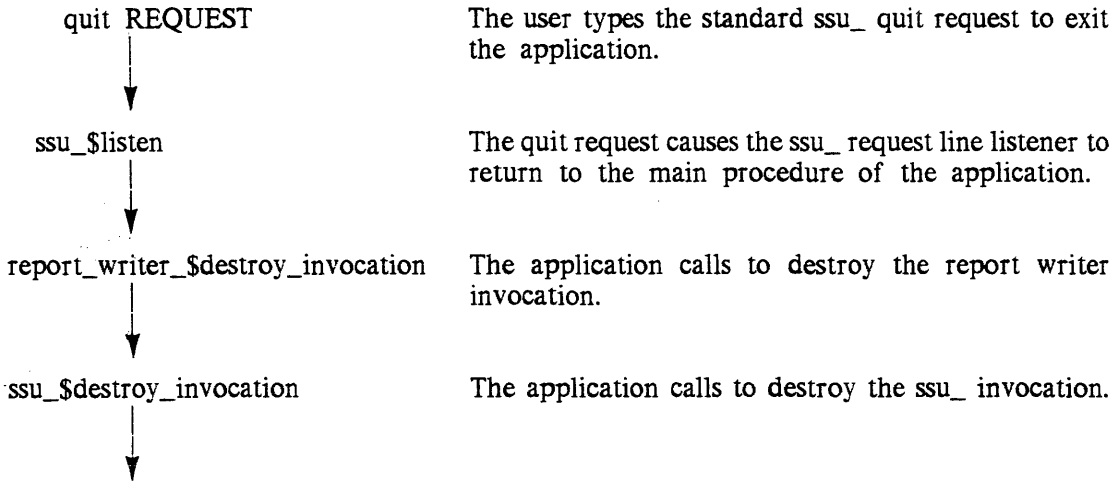


Figure 3-4. Destroying an Invocation

The last step in using report_writer_ is to call the destroy_invocation entrypoint. This entrypoint takes the report_writer_ info pointer as its only parameter. If the pointer is null, the call is ignored. The following code fragment illustrates its usage.

```
call report_writer_$destroy_invocation (report_writer_info_ptr);
```

SECTION 4

REPORT WRITER REQUEST DESCRIPTIONS

This section contains a description of the MRW requests. Each request description contains the name (including the abbreviated form, if any), discusses its purpose, and shows correct usage. Notes and examples are included where necessary for clarity.

The following list summarizes the MRW requests.

column_value, clv
: returns the value of the specified column for the current row, previous row, or next row.

display, di
: retrieves selected data, creates a report, and displays the information or writes it to a file.

display_builtins, dib
: returns the current values for requested built-ins.

list_format_options, lsfo
: lists the names and values of format options.

restore_format_options, rsfo
: restores saved report layouts.

save_format_options, svfo
: saves current values of format options for future use.

set_format_options, sfo
: changes/sets report format options.

The remainder of this section contains a detailed description of each request.

Request: column_value, clv

This request returns the value of the specified column for the current row, previous row, or next row. It can only be used as an active request. It is used within a formatted report produced by the display request to obtain the value of a column. It is an error to use this request anywhere except in a header/footer or editing string within a report produced by the display request.

USAGE AS AN ACTIVE REQUEST

```
[clv column_id {-control_args}]
```

where:

1. column_id
   specifies which column value is to be returned. It can be given as the name of the column or the number of the column as selected through the subsystem.

2. control_args
   can be chosen from the following:

   -current_row, -crw
   returns the value of the named column for the current row. (Default)

   -default STR
   returns the character string STR when there is no previous row, or when there is no next row. (Default value for STR is "" if this control argument is not provided.)

   -next_row, -nrw
   returns the value of the named column for the next row. If there is no next row, the string "" is returned unless changed by the -default control argument.

   -previous_row, -prw
   returns the value of the named column for the previous row. If there is no previous row, the string "" is returned unless changed by the -default control argument.

NOTE

When a subtotal is being generated, the column_value request returns the value of the subtotal, rather than the value of the column. An editing string for a column like "[pic $99v.99 [clv salary]]", would edit the value of the salary column through the picture active request for every row. When a subtotal is being generated, the value of the salary subtotal is edited through the picture active request. This behavior also applies to subcounts, counts, and totals, in addition to subtotals.

EXAMPLES

```
[clv foo]

[clv 3]
```

```
[clv foo -previous_row]

[clv foo -next_row -default NULL]
```

Request: display, di

This request retrieves selected data, creates a report, and displays it on the terminal or sends it to a file or an io switch.

USAGE

```
di {-control_args}
```

where control_args can be chosen from:

Note: The following list identifies all control arguments grouped by function.

CONTROLLING WARNING MESSAGES
- -brief
- -long

DISPLAYING PAGES AND PORTIONS OF PAGES
- -all
- -character_positions
- -page

DATA RETRIEVAL INITIATION AND TERMINATION
- -discard_retrieval
- -keep_retrieval
- -new_retrieval
- -old_retrieval

REPORT INITIATION AND TERMINATION
- -discard_report
- -keep_report
- -new_report
- -old_report

SORTING RETRIEVED DATA
- -sort

CONTROLLING REPORT OUTPUT
- -extend
- -output_file
- -output_switch
- -truncate

VIDEO SYSTEM SCROLLING FUNCTIONS
-enable_escape_keys
-enable_function_keys
-scroll
-set_key
-window

MULTI-PASS REPORT FORMATTING
-passes

TEMPORARY STORAGE SPECIFICATION
-temp_dir

-all, -a
displays all pages of the report. This argument is incompatible with the -pages control argument. (Default)

-brief, -bf
suppresses warning messages.

-character_positions STR1 {STR2}, -chpsn STR1 {STR2}
where STR1 and STR2 define the left and right character positions of a vertical section of the report. STR1 must be given and defines the left margin position to begin from. STR2 is optional, and if not given, defaults to the rightmost character position of the report. If this control argument is not given, the entire page is displayed.

-discard_report, -dsrp
deletes the report on termination. (Default)

-discard_retrieval, -dsr
deletes retrieved data on termination. (Default)

-enable_escape_keys, -eek
specifies the use of escape key sequences, rather than the function keys and arrow keys on the terminal, for scrolling functions. This is the default if the -scroll control argument is given and the terminal does not have the necessary set of function keys and arrow keys (see -enable_function_keys). (In the following description, the mnemonic "esc-" means the escape key on the terminal.) The following escape key sequences are used if this control argument is given, or the terminal lacks the necessary set of keys:

FUNCTION NAME	KEY SEQUENCE
forward	esc-f
backward	esc-b
left	esc-l
right	esc-r
help	esc-?
set_key	esc-k
set_scroll_increment	esc-i
quit	esc-q
redisplay	esc-d

start_of_report	esc-s
end_of_report	esc-e
multics_mode	esc-m
goto	esc-g

-enable_function_keys, -efk
specifies the use of terminal function keys and arrow keys for scrolling functions. This is the default when the -scroll control argument is given and the terminal has at least nine function keys and four arrow keys. (In the following description, the mnemonic fN means function key N, where N is the number of the function key. The mnemonic down_arrow means the down arrow key, up_arrow means the up arrow key, left_arrow means the left arrow key, and right_arrow means the right arrow key. The following key sequences are used if this control argument is given and the terminal has the necessary set of keys:

FUNCTION NAME	KEY SEQUENCE
forward	down_arrow
backward	up_arrow
left	left_arrow
right	right_arrow
help	f1 (function key)
set_key	f2
set_scroll_increment	f3
quit	f4
redisplay	f5
start_of_report	f6
end_of_report	f7
multics_mode	f8
goto	f9

-extend
appends the report to an existing file rather than replacing it if the -output_file control argument is used. (Default is to truncate an existing file if this control argument is not provided.)

-keep_report, -krp
keeps the report on termination. This control argument is necessary in order to use -old_report on subsequent invocations of display.

-keep_retrieval, -kr
keeps retrieved data to allow re-use on subsequent invocations of the display request. Previously retrieved sorted data retains the sort order.

-iong, -ig
displays warning messages when a control argument such as -old_retrieval is used and the data from a previous retrieval is not available. (Default)

-new_report, -nrp
creates a new report. (Default)

-new_retrieval, -nr
begins a new retrieval. (Default)

-old_report, -orp
uses the report created in the previous invocation. Use of this control argument requires that -keep_report be used in the prior invocation of display.

-old_retrieval, -or
uses data retrieved during the previous invocation. Use of this control argument requires that -keep_retrieval be used in the prior invocation of display.

-output_file path, -of path
where path is the name of the file which contains the formatted report. If this control argument or -output_switch is not given, the report is displayed on the terminal. This argument is incompatible with the -output_switch control argument.

-output_switch switch_name, -osw switch_name
where switch_name is the name of a switch to be used to display the report. If this control argument or -output_file is not given, the report is displayed on the terminal. It is an error to use this control argument if the named switch is not already open and attached when display is invoked. This argument is incompatible with the -output_file control argument.

-page STR, -pg STR; -pages STR, -pgs STR
where STR is a blank-separated list of pages (N N) or comma-separated page ranges where STR is a blank-separated list of pages (N N) or comma-separated page ranges (N,N). Page ranges can also be given as N, or "N,$" which means from page N to the end of the report, or simply $ which means the last page. This argument is incompatible with the -all control argument.

-passes N, -pass N
where N is the number of times the report is to be formatted. No output is produced until the last formatting pass of the report. (Default value for N is 1)

-scroll
specifies scrolling the report according to key sequences read from the terminal. Only terminals supported by the Multics video system can use the scrolling feature. If the -window control argument is not used, the -scroll argument creates a uniquely named window for the display of the report. The user_i/o window is reduced to four lines and the remaining lines are used for the uniquely named report display window. The minimum size for this window is five lines, so the user_i/o window must be at least nine lines before invoking display, unless the -window control argument is used.

-set_key STR, -sk STR; -set_keys STR, -sks STR
specifies that the named scrolling functions are to be set to the provided key sequences. STR is a blank-separated list of one or more scrolling function names and key sequences, given as:

```
function_name key_sequence ... {function_name  key-sequence}
```

The function names can be chosen from the set described under -enable_escape_keys or -enable_function_keys control arguments. The key sequences can be given as the actual sequences or mnemonic key sequences. The provided mnemonics can be:

fN	where N is the number of the desired function key
esc- or escape-	corresponds to the escape character
ctl-x or control-x	correspods to the character sequence generated when the control key is held while also pressing the character named by "x"
down_arrow	corresponds to the down arrow key
up_arrow	corresponds to the up arrow key
left_arrow	corresponds to the right arrow key
home	corresponds to the home key

-sort STRs {-ascending | -descending} {-case_sensitive|
-non_case_sensitive), -sort STRs {asc | -dsc} -cs | -ncs}
where STRs are the names of columms or numbers corresponding to the position of the columns as selected through the subsystem. It can be followed by -ascending or -descending, and -case_sensitive or -non_case_sensitive. (Default is -ascending and -case_sensitive.)

-temp_dir dir_name, -td dir_name
specifies that the given directory be used for storing the retrieved data, saving the report if -keep_report is used, and sorting workspace if -sort is used instead of the process directory. This temporary directory continues to be used until another new temporary directory is requested. A new temporary directory can only be specified when a new retrieval and new report are requested.

-truncate, -tc
replaces the contents of the existing file if the -output_file control argument is used. (Default is to truncate if the -extend control argument is not provided.,

-window STR, -win STR
specifies that the window named by STR be used for the display of the report. This argument is only meaningful when the -scroll argument is also used. If this control argument is used, the window named by STR must be attached and open under the video system, and it must be at least five lines high.

EXAMPLES

```
di

di -output_file foo

di -keep_retrieval -sort bar -descending -non_case_sensitive

di -keep_retrieval -keep_report -of fool -character_positions 1 132
```

```
di -old_retrieval -old_report -of foo2 -character_positions 133 260

di -pages 1 3 12,19 58,$ -output_switch foo

di -sort foo -descending bar -non_case_sensitive
```

Request: display_builtins, dib

This request returns the current value of the built-in named by STR. It can only be used as an active request. It is used within a formatted report produced by the display request to obtain the current value of the specified built-in. It is an error to use this request anywhere except in a header/footer or editing string within a report produced by the display request.

USAGE AS AN ACTIVE REQUEST

```
[dib STR]
```

where STR can be any one of the following built-ins:

current_pass_number
: the number of the current pass. The number begins with 1 and is incremented by 1 for each additional formatting pass over the report.

current_row_number
: the number of the current row of the report.

first_row
: true if the current row is the first row of the report, or false if it is not the first row of the report.

last_page_number
: the number of the last page of the report, or "0" if it is the first pass over the report. After each formatting pass, the number is updated with the number of the last page.

last_pass
: true if this is the last formatting pass of the report, or false if this is not the last pass of the report.

last_row
: true if the current row is the last row of the report, or false if the current row is not the last row of the report.

last_row_number
: the number of the last row of the table, or "0" if it is the first pass over the report. After the first formatting pass the number is set to the number of the last row.

page_number
: the number of the current page of the report.

previously_processed_row
true if the current row was processed on the preceding page but the row value would not fit and had to be deferred to the current page, or false if this is the first time the current row is being processed.

Request: list_format_options, lsfo

This request lists the names and values of individual report formatting options, all report formatting options, or the active report formatting options. As an active request, it returns the value of the single specified format option.

USAGE

```
lsfo -control_arg  -OR-  lsfo -format_option_args
```

USAGE AS AN ACTIVE REQUEST

```
[lsfo -format_option_arg]
```

where:

1. control_args
   can be chosen from the following:

   -active, -act
   specifies that only the active formatting options are to be listed. (Default) "help formatting_options.gi" is typed for more information on active formatting options. This control argument is incompatible with -all and the format option arguments. If -active and -all are both given, the last one supplied is used.

   -all, -a
   specifies that all formatting options are to be listed. This control argument is incompatible with -active and the format option arguments. If -all and -active are both given, the last one supplied is used.

2. format_option_args
   can be one or more of the following:

   GENERAL REPORT OPTIONS

   -delimiter, -dm
   displays the character used to delimit the different portions of a header or footer.

   -format_document_controls, -fdc
   displays the interpretation of embedded format document controls when filling (on), or the treatment of embedded controls as ordinary text (off).

-hyphenation, -hph
displays hyphenation where possible for overlength values (on), or no hyphenation (off).

-page_footer_value, -pfv
displays the page footer placed at the bottom of each page.

-page_header_value, -phv
displays the page header placed at the top of each page.

-page_length, -pl
displays the length of each formatted page given as the number of lines.

-page_width, -pw
displays the width of each formatted page given as the number of character positions.

-title_line, -tl
displays printing of the title line (on) or the suppression of the title line (off).

-truncation, -tc
displays the character or characters used to indicate truncation.

GENERAL COLUMN OPTIONS

-column_order. -co
displays the order of columns in the detail line.

-count, -ct
displays the columns which have counts taken on them.

-exclude, -ex
displays the columns to be excluded in the detail line.

-group, -gr
displays the columns used to group a number of rows based on their values.

-group_footer_trigger, -gft
displays the columns which can cause the generation of the group footer.

-group_footer_value, -gfv
displays the group footer placed after each group of rows.

-group_header_trigger, -ght
displays the columns which can cause the generation of the group header.

-group_header_value, -ghv
displays the group header placed before each group of rows.

-outline, -out
displays the columns which can duplicate suppression.

-page_break, -pb
displays the columns which can cause a break to a new page.

-row_footer_value, -rfv
displays the row footer placed after each row value.

-row_header_value, -rhv
displays the row header placed before each row value.

-subcount, -sct
displays the columns that have subcounts taken on them.

-subtotal, -stt
displays the columns that have subtotals taken on them.

-total, -tt
displays the columns that have totals taken on them.

SPECIFIC COLUMN OPTIONS

In the following descriptions, column_id means the column name, the number of the column selected through the subsystem, or a star name which is matched against the column names.

-alignment column_id, -al column_id
displays the alignment mode within the display width for the specified column.

-editing column_id, -ed column_id
displays the editing string for the specified column.

-folding column_id, -fold column_id
displays the folding action taken when the column value exceeds the display width for the specified column.

-separator column_id, sep column_id
displays the character string that separates the specified column from the column in the detail line which immediately follows it.

-title column_id, -ttl column_id
displays the character string that is placed at the top of the page above the specified column.

-width column_id, -wid column_id
displays the display width in the detail line for the specified column.

NOTES

Refer to the description of the set_format_options request for a complete list of the default values for the format options and a discussion of their allowed values. When used as an active request, only one format_option_arg can be specified.

EXAMPLES

```
lsfo

lsfo -all

lsfo -width 1 -alignment salary

lsfo -page_width -title ** -page_length
```

Request: restore_format_options, rsfo

This request restores the saved report layout specified by path. Only the formatting options found in the saved report layout have their values changed.

USAGE

```
rsfo path
```

where path is the pathname of the saved report format to be restored. If path does not contain an fo.ec_name suffix, one is assumed. The ec_name is specific to the subsystem which is using the report writer, and the ec suffix for the subsystem is substituted here by the report writer.

NOTES

Refer to the save_format_options request for detail on the content of the saved report format.

EXAMPLES

```
rsfo sample_display_format
rsfo another_display_format.fo.lec
```

Request: save_format_options, svfo

This request saves the current values of format options as a subsystem exec_com. The saved format can be restored with the restore_format_options request. The file is saved by utilizing a suffix of fo.ec_name where ec_name is substituted by the subsystem's ec suffix by the report writer. Individual format options, active format options, or all of the format options can be saved. The requests used to select the set of columns through the subsystem can also be saved.

USAGE

```
svfo path {-format_option_args} {-control_args}
```

where:

1. path is the pathname of the segment that contains the saved format. If path does not have an fo.ec_name suffix, one is assumed.

2. format_option_args
   refer to the set_format_options request for a complete description of the format option arguments. Each format option named has its value saved in the exec_com specified by path. These arguments are incompatible with the -all and -active control arguments.

GENERAL REPORT OPTIONS

-delimiter, -dm
-format_document_controls, -fdc
-hyphenation, -hph
-page_footer_value, -pfv
-page_header_value, -phv
-page_length, -pl
-page_width, -pw
-title_line, -tl
-truncation, -tc

GENERAL COLUMN OPTIONS

-column_order, -co
-count, -ct
-exclude, -ex
-group, -gr
-group_footer_trigger, -gft
-group_footer_value, -gfv
-group_header_trigger, -ght
-group_header_value, -ghv
-outline, -out
-page_break, -pb
-row_footer_value, -rfv
-row_header_value, -rhv
-subcount, -sct
-subtotal, -stt
-total, -tt

SPECIFIC COLUMN OPTIONS

-alignment, -al
-editing, -ed
-folding, -fold
-separator, -sep
-title, -ttl
-width, -wid

3. control_args
can be one or more of the following:

-active, -act
specifies that only the active formatting options are to be saved. (Default) "help formatting_options.gi" is typed for more information on active formatting options. This control argument is incompatible with the format option arguments and the -all control argument. If -active and -all are given, the last one supplied is used. (Default)

-all, -a
specifies that all formatting options are to be saved. This control argument is incompatible with the format option arguments and the -active control argument. If -all and -active are given, the last one supplied is used.

-query
specifies that the subsystem query request used to select the columns is to be saved. A restore_format_options on the saved format also restores and makes the saved query current.

EXAMPLES

```
svfo report_layout

svfo report_layout -all

svfo report_layout -query

svfo report_layout -page_header_value -page_footer_value

svfo report_layout -page_header_value -width salary

svfo report_layout -width ** -page_footer_value
```

Request: set_format_options, sfo

This request sets individual report format options to user-specified or default values, and/or all formatting options to default values.

USAGE

```
sfo {-format_option_args} {-control_args}
```

Note: The option value given for any format option argument can be the control arguments -default or -prompt. If -default is given for the value, report writer sets the value of the format option to the system default. If -prompt is given for the value, report writer prompts for the value with the prompt string "Enter FORMAT_OPTION_NAME.". A line consisting of the single character "." terminates the prompted input mode. To suppress display of the prompt string, use the -brief control argument.

where:

1. format_option_args
can be one or more of the following:

GENERAL REPORT OPTIONS

-delimiter CHAR, -dm CHAR
CHAR is the character used to delimit the different portions of a header or footer and can be set to any printable character. (Default value for CHAR is !)

-format_document_controls STR, -fdc STR
STR determines if the format_document_ subroutine is to interpret format document control lines when filling overlength text. STR can be set to on, meaning format_document_ interprets control lines in the text and provides special filling actions based on the embedded control lines. (Default value for STR is off, meaning format_document_ does not check for control lines embedded in text.)

-hyphenation STR, -hph STR
the value of -hyphenation determines if hyphenation is to be attempted when filling overlength character strings. STR can be set to "on," specifying that hyphenation is to be attempted. (Default value for STR is off, meaning no hyphenation is attempted.)

-page_footer_value STR, -pfv STR
STR is the page footer placed at the bottom of each page. The page footer can consist of more than one line, and each line can have a left, right, and center portion. The individual portions of each line are delimited by the delimiter character. Active requests found in the footer are evaluated and their return value is placed into the footer before folding and alignment takes place. Portions of a footer with zero length have their space on the page redistributed to the other portions whose lengths are not zero. For example, if the page footer contained only a center portion:

```
!!Sample Center Portion!!
```

the text is centered on the page and has the full page width available for the text. Similarly, a left portion or right portion only is aligned to the left or right of the page and has the full page width available for placement of text. Two exceptions to this action are when the footer has a left, right, and center portion, and the left or right portion has a zero length, such as:

```
!left part!center part!!
```

or

```
!!center part!right part!
```

in which case the left or right part of the page is unavailable for placement of text (i.e., the space is not redistributed to the other two portions). If the redistribution of the available page width is not desired, the placement of a single blank into a portion such as "!<SP>!Center Part!<SP>!" prevents the redistribution from taking place because each portion has a length greater than zero. (Default value for STR is "", meaning there is no page footer provided by default.)

-page_header_value STR, -phv STR
STR is the page header placed at the top of each page. Refer to the description of -page_footer_value for the content of a header. (Default value for STR is "", meaning there is no page header provided by default.)

-page_length N, -pl N
N is the length of each formatted page given as number of lines. N can be given as "0" or any positive integer. 0 means the report is not to be paginated and is created as one continuous stream. (Default value for N is 0)

-page_width N, -pw N
N is the width of each formatted page given as the number of character positions. N can be given as "0" or any positive integer. 0 means the page_width is always set by report writer to be the exact width needed to contain all of the columns specified in the query. If N is greater than zero and the width for any column exceeds N, the width of the column is automatically set to N. (Default value for N is 79)

-title_line STR, -tl STR
STR determines if a title line is to be printed. STR can be set to "off" to inhibit the printing of the title line. (Default value of STR is on, meaning a title line is printed at the top of each page.)

-truncation STR, -tc STR
STR determines the character(s) to be used to indicate truncation of some value. STR can be set to any sequence of printable characters. (Default value for STR is *)

GENERAL COLUMN OPTIONS

-column_order column_list, -co column_list
column_list determines the order in which columns appear in the detail line. column_list can be set to a list of column names or numbers. Columns missing from this list are placed after the columns which appear in the list. That is, if five columns were selected and the column_order value is given as "3 2", the complete order would be "3 2 1 4 5". (Default value for column_list is the list of columns from the query, in the order supplied, meaning that the columns appear in the exact order as they appear in the query.)

-count column_list, -ct column_list
column_list determines the columns for which counts are generated. column_list can be set to a list of column names or numbers. Counts are generated after the last detail line. If a count is requested on a column that is excluded, the count is also excluded from the page. An exception to this rule is when all columns are excluded. Counts are provided in this case to allow reports consisting of some combination of counts, subcounts, totals, and subtotals only. (Default value for column_list is "", meaning no columns have counts generated.)

-exclude column_list, -ex column_list
column_list determines if any of the columns selected in the query are excluded from the detail line. column_list can be set to a list of column names or numbers. (Default value for column_list is "", meaning no columns are excluded.)

-group column_list, -gr column_list
column_list determines the grouping of a number of rows based on the values of one or more columns. column_list can be set to a list of column names or numbers. The column or columns named in the list become a hierarchy of columns. The first column named is the major column, and the last column named becomes the minor column. The hierarchy of columns can be used with the outline, page_break, and subtotal options described below. (Default value for column_list is "", meaning no group of rows is defined.)

-group_footer_trigger column_list, -gft column_list
column_list determines when to generate the group footer. column_list can be set

to a list of column names or numbers. The columns which appear in this list must also appear in the column list associated with the -group option. If the -group option is set to a new value, columns which are eliminated from the column_list are also eliminated from the -group_footer_trigger column_list. When any of the columns specified in the column_list are about to change with the next row, the group footer is evaluated. The group footer is always evaluated after the last row of the report. (Default value for column_list is "", meaning no group footer triggers are defined.)

-group_footer_value STR, -gfv STR
STR is the group footer placed after each group of rows when any of the columns associated with the -group_footer_trigger option changes. Refer to the description of -page_footer_value above for the content of a header/footer. (Default value for STR is "", meaning there is no group footer defined.)

-group_header_trigger column_list, -ght column_list
column_list determines when to generate the group header. column_list can be set to a list of column names or numbers. The columns which appear in this list must also appear in the column list associated with the -group option. If the -group option is set to a new value, columns which are eliminated from the column_list are also eliminated from the -group_header_trigger column_list. When any of the columns specified in the column_list have just changed with the current row, the group header is evaluated. The group header is always evaluated before the first row of the report. (Default value for column_list is "", meaning no group header triggers are defined.)

-group_header_value STR, -ghv STR
STR is the group header placed before each group of rows when any of the columns associated with the -group_header_trigger option changes. Refer to the description of -page_footer_value above for the content of a header/footer. (Default value for STR is "", meaning there is no group header defined.)

-outline column_list, -out column_list
column_list determines if duplicate values in a column are to be suppressed. column_list can be set to a list of column names or numbers. If the value of a named column is the same as its previous value, then the value is suppressed unless it is the first line of a new page. (Default value for column_list is "", meaning no columns have duplicate values suppressed.)

If any of the named columns are a member of the "group" of rows defined by the group option, then it, and all of the columns more major in this group, are outlined. A change in value of any one column displays all columns lower in the hierarchy in addition to the column that changed. An exception is the first line on a new page, in which case duplicate values are never suppressed.

-page_break column_list, -pb column_list
column_list determines when page breaks are generated. column_list can be set to a list of column names or numbers. The columns specified in the list are examined, and when their values change, a new page break is generated. If any of the named columns are a member of the "group" of rows defined via the group option, then it, and all columns more major in the group, are examined for page breaks. (Default value for column_list is "", meaning that no columns are examined for page breaks.)

-row_footer_value STR, -rfv STR
STR is the row footer placed after each detail line. Refer to the description of -page_footer_value (above) for the content of a footer. (Default value for STR is "", meaning that no row footer is provided.)

-row_header_value STR, -rhv STR
STR is the row header placed before each detail line. Refer to the description of -page_footer_value (above) for the content of a header. (Default value for STR is "", meaning that no row header is provided.)

-subcount subcount_spec, -sct subcount_spec
subcount_spec determines what columns subcounts to generate, when they should be generated, and what type of subcount is generated. (Default value for subcount_spec is "", meaning that no subcounts are generated for any columns.)

subcount_spec can consist of one or more blank-separated "triplets." The syntax of a triplet is:

```
column_1,column_2{reset | running}
```

where:

column_1
is the name or number of the column for which a subcount is generated.

column_2
is the name or number of a column whose value is examined to determine when to generate the subcount. When the value of the column being examined changes, the subcount is generated. If this column is a member of the group of rows defined via the group option, it, and all columns more major in the group, are examined for subcount generation.

reset | running
indicates the type of subcount desired. If reset is selected, the subcount counter is reset to 0 each time a subcount is generated. If running is selected, the subcount is not reset to 0. If a subcount is requested on a column that is excluded, the subcount is also excluded from the page. An exception to this rule is when all columns are excluded. Subcounts are provided in this case to allow reports consisting of some combination of counts, subcounts, totals, and subtotals only. (Default is reset)

-subtotal subtotal_spec, -stt subtotal_spec
subtotal_spec determines what column subtotals to generate, when they should be generated, and what type of subtotal is generated. (Default value for subtotal_spec is "", meaning no subtotals are generated for any columns.)

subtotal_spec can consist of one or more blank-separated triplets. The syntax of a triplet is:

```
column_1,column_2{,reset | running}
```

where:

column_1
is the name or number of the column for which a subtotal is generated.

column_2
is the name or number of a column whose value is examined to determine when to generate the subtotal. When the value of the column being examined changes, the subtotal is generated. If this column is a member of the group of rows defined via the group option, it, and all columns more major in the group, are examined for subtotal generation.

reset | running
indicates the type of subtotal desired. If reset is selected, the subtotal counter is reset to 0 each time a subtotal is generated. If running is selected, the subtotal is not reset to 0. If a subtotal is requested on a column that is excluded, the subtotal is also excluded from the page. An exception to this rule is when all columns are excluded. Subtotals are provided in this case to allow reports consisting of some combination of counts, subcounts, totals, and subtotals only. (Default is reset)

-total column_list, -tt column_list
column_list determines what column totals to generate. (Default value for column_list is "", meaning no totals are generated for any columns.)

column_list can be set to a list of column names or numbers. Totals are generated after the last detail line. If a total is requested on a column that is excluded, the total is also excluded from the page. An exception to this rule is when all columns are excluded. Totals are provided in this case to allow reports consisting of some combination of counts, subcounts, totals, and subtotals only.

SPECIFIC COLUMN OPTIONS

In the following descriptions, column_id means the column name, the number of the column in the query, or a star name which is used to match column names.

-alignment column_id STR, -al column_id STR
column_id specifies which column the alignment applies to and STR is the alignment mode. STR can be set to center, left, right, both, or decimal N. The default value for STR depends upon the type of column selected. Character and bit strings default to left-alignment, decimal data with a non-zero scale defaults to decimal-point-alignment, and all other types default to right alignment. For decimal-point-alignment, the decimal alignment position within the display width is given a default value. This alignment position can be changed by specifying the value as "decimal N", where N is the character position within the display width where the decimal point is aligned. The alignment mode "both" specifies that the column value is aligned to the leftmost and rightmost character positions within its display width. Text is padded by insertion of uniformly distributed whitespace if necessary.

-editing column_id STR, -ed column_id STR
STR specifies the additional editing to be done to the column value before it is placed on the page and column_id specifies which column the editing applies to.

Multics active functions and subsystem active requests are normally used to provide additional editing. For example, the editing value:

```
[e pic $99,999v.99 [column_value salary]]
```

places commas and dollar signs in the salary column. (Default value for STR is "", meaning additional editing is not done.)

Refer to the column_value request for a description of usage.

-folding column_id STR, -fold column_id STR
STR determines what type of action occurs when a column value exceeds its display width and column_id specifies which column the folding applies to. STR set to truncate means that the value of the column is truncated to fit in the display width and the truncation character(s) are placed at the end of the value to indicate that truncation occurred. (Default value for STR is fill, meaning portions of the value which exceed the display width are moved down to the next line(s) until a correct fit is obtained.)

-separator column_id STR, -sep column_id STR
STR separates a column from the next one following it and column_id specifies which column the separator applies to. The last column on a line does not have a separator. STR can be any sequence of printable characters. (Default value for STR is "<SP><SP>")

-title column_id STR, -ttl column_id STR
STR is the title placed above the column at the start of each page if the title_line option is set "on" and column_id specifies which column the title applies to. (Default value of STR is the name of the column. If the title is not the same number of characters as the display width of the column, the title is centered within the display width for its associated column. If the value of title is wider than the display width of the column, it is filled or truncated to obtain a correct fit, depending on the folding action of the parent column.)

-width column_id N, -wid column_id N
N determines the display width for a column and column_id specifies which column the width applies to. N can be set to any positive integer. (Default value for N is the number of character positions needed to contain the value, after conversion from the subsystems data type, to character format.)

2. control_args
can be chosen from the following:

-brief, -bf
specifies that the prompt string for values is not to be displayed. If the -brief and -long control arguments are both entered on the request line, the last one supplied is used.

-default
specifies that report writer set the value of the format option which immediately precedes this control argument to the system-supplied default.

-long, -lg
displays "Enter FORMAT_OPTION_NAME" prompt string for values when the -prompt control argument is provided. (Default) If the -brief and -long control arguments are both entered in the request line, the last one supplied is used.

-no_reset, -nrs
specifies that formatting options are not to be reset to system default values. (Default is that only user-specified options can be changed.) If the -reset and -no_reset control argument are both entered in the request line, the last one supplied is used.

-prompt
specifies that report writer prompts for the value of the format option which immediately precedes this control argument. A prompt string is written before the prompting action unless the -brief control argument is used. A line consisting of the single character "." terminates the prompted input mode.

-reset, -rs
specifies that all formatting options are to be reset to system default values before the values are changed for any other format options specified in the request line. If -reset and -no_reset are both entered in the request line, the last one supplied is used.

-string STR, -str STR
enters STR as a format option value when STR begins with a hyphen.

NOTES

At least one format option argument or the -reset control argument must be specified. Format option arguments and control arguments can be mixed freely in the request line, but a control argument cannot be placed between a format option name and a format option value. For example:

```
sfo -page_width 80 -reset
```

is a valid request, but

```
sfo -page_width -reset 80
```

is not valid. If a value is to be set that begins with a hyphen, the -string control argument must be given before the value, to distinguish it from control arguments and format option arguments.

EXAMPLE

```
sfo -width 1 25

sfo -title emp_name "Employee Name"

sfo -reset -page_width 80 -page_length 60

sfo -page_footer_value "!!-[display_builtins page_number]-!!"
```

```
sfo -page_header_value -prompt
Enter page_header_value.
![execute date]!SAMPLE REPORT![execute time]!
!!!!
!!--Page [display_builtins page_number]--!!
.

sfo -exclude exchange extension -width area_code 12

sfo -editing area_code "[e format_line ^a/^a-^a [column_value area_code]
   [column_value exchange] [column_value extension]]"
```

SECTION 5

REPORT WRITER SUBROUTINE DESCRIPTION

This section contains the description of the report_writer_ subroutine and its functions, presented in alphabetic order. The description contains the name of the subroutine, discusses the purpose of the subroutine, lists the entry points, and describes the correct usage for each entry point. Notes are included for clarity.

Name: report_writer_

The report_writer_ subroutine provides a programmer interface to the MRW system. Through it, application subsystems can include generalized report writing capabilities, with a minimal amount of coding required by the subsystem developer. The following entrypoints are provided:

convert_and_move_row
: called to have a row value converted from differing data types and moved into the row value buffer.

create_invocation
: called to create a report_writer_ invocation.

define_columns
: called to inform report_writer_ that a set of columns are available for usage.

destroy_invocation
: called to destroy a report_writer_ invocation.

set_report_writer_info_ptr
: called to have the report_writer_ standard requests use the supplied report_writer_ info pointer (for subsystems that have more than one report_writer_ invocation present in the one subsystem (ssu_) invocation).

set_table_manager
: called to have report_writer_ use the specified table manager procedure for this report_writer_ invocation.

Entry: convert_and_move_row

This entry converts a row of differing data types to a row of character string values and places it in the row value buffer.

USAGE

```
declare report_writer_$convert_and_move_row entry (ptr, (*) ptr);
call report_writer_$convert_and_move_row (report_writer_info_ptr,
   value_ptrs);
```

ARGUMENTS

report_writer_info_ptr
: is the pointer to the report_writer_ info structure returned by the create_invocation entrypoint. (Input)

value_ptrs
: is an array of pointers that point to the individual column values that are to be converted. (Input)

NOTES

The number of pointers in the array must be equivalent to the number of columns defined in the table. Refer to "Data Conversions" in Section 3 for a code fragment which shows the usage of this subroutine.

Entry: create_invocation

This entry creates a report_writer_ invocation. It adds the standard report_writer_ request table and the standard requests info segments directory at location 99999 (see Notes).

USAGE

```
declare report_writer_$create_invocation entry (char(*), ptr, ptr,
        fixed bin(35), char(*) varying);
call report_writer_$create_invocation (table_manager_name,
     sci_ptr, report_writer_ptr, code, message);
```

ARGUMENTS

table_manager_name
: is the name of the table manager procedure for the application subsystem that is creating the report writer invocation. If the name is blank, report_writer_ obtains the name of the application subsystem via the ssu_$get_subsystem_name subroutine and uses this as the table manager procedure name. Entry variables are constructed by report_writer_ using the name of the table manager procedure. (Input)

sci_ptr
: is the ssu_ info pointer returned to the application subsystem when the ssu_ invocation is created. If this parameter is null, the report_writer_ invocation is not created. (Input)

report_writer_info_ptr
: is the pointer to the report_writer_ info structure. This pointer is passed as the first parameter to every other report_writer_ entry. (Output)

code
: is a standard error code. If this is non-zero the invocation could not be created. (Output)

message
: if code is non-zero, this parameter contains the reason why the invocation could not be created. Declaring this argument as "character(128) varying" provides an area large enough to contain any returned error message. (Output)

NOTES

The table manager procedure is a program supplied by the application subsystem to manage data retrieval from its source data file. The report_writer_ creates entry variables of the form "table_manager_name$XX", where XX are the entries that must be available in the table manager

procedure. The required entries are create_table, delete_table, and get_row. The optional entry is get_query. See "Subsystem Table Manager Procedure" in Section 3 for additional information.

The report_writer_ request tables provide a mechanism to include all of the report_writer_ standard requests (the default when an invocation is created), or selected individual report_writer_ requests. The request table "report_writer_request_table_$standard_requests" is used by the report_writer_$create_invocation entrypoint in a call to ssu_$add_request_table to add the report_writer_ standard requests to the subsystem's set of requests. If the standard requests are not required, the subsystem should call ssu_$delete_request table with this as a parameter, after the call to report_writer_$create_invocation. The report_writer_ standard requests are: column_value, display, display_builtins, list_format_options, restore_format_options, save_format_options, and set_format_options. The other request tables defined to allow the addition of single requests are report_writer_request_table_$XXX_request, where XXX is the short name of the request. These can be chosen from: clv, di, dib, lsfo, rsfo, sfo, and svfo.

The report_writer_info_dirs_ data segment provides a mechanism to include the directory containing the info segment for a request. The report_writer_info_dirs_$standard_requests is used by the report_writer_$create_invocation entrypoint in a call to ssu_$add_info_dir, to add the info directory containing the info segments for the standard requests. If the standard requests are not required, the subsystem should call ssu_$delete_info_dir with this as a parameter, after the call to report_writer_$create_invocation. The other data items defined to allow the addition of directories which contain the info segment for a single request are report_writer_info_dirs_$XXX_requests, where XXX is the short name of the request. These can be chosen from: clv, di, dib, lsfo, rsfo, sfo, and svfo.

Entry: define_columns

This entry is used to inform report_writer_ that a set of columns have been selected. All specific and general column options have new values assigned based on the newly selected columns.

USAGE

```
declare report_writer_$define_columns entry (ptr, ptr,
        fixed bin(35), char(*) varying);
call report_writer_$define_columns (report_writer_info_ptr,
     row_info_ptr, code, message);
```

ARGUMENTS

report_writer_info_ptr
: is the pointer to the report_writer_ info structure returned by the create_invocation entrypoint. (Input)

row_info_ptr
: is a pointer to the row_info structure described in the include file rw_row_info.incl.pl1 (See "Notes" below.) If this pointer is null, any previously defined specific and general column option values are deleted, and the only operations allowed are to general report options until this entry is called again with a valid row_info pointer. (Input)

code
is a standard error code. If this code is non-zero the values of any previously defined specific and general columns options will remain unchanged. (Output)

message
if code is non-zero this parameter will contain the reason for the failure. Declaring this argument as "character(128) varying" provides an area large enough to contain any returned error message. (Output)

NOTES

The row_info structure declared in the include file rw_row_info.incl.pl1 is listed below, followed by a description of the members of this structure.

```
declare 1 row_info aligned based (row_info_ptr),
        2 version char(8) unaligned,
        2 value_ptr ptr,
        2 value_length fixed bin(21),
        2 number_of_columns fixed bin,
        2 current_column_number fixed bin,
        2 column (row_info_init_number_of_columns
                  refer (row_info.number_of_columns)),
          3 names char(128) varying,
          3 descriptors bit(36),
          3 lengths fixed bin(21),
          3 indexes fixed bin(21);
```

where:

version
specifies to report_writer_ that the application subsystem is using a particular version of the structure. The only version currently supported by report_writer_ is version 1. The value of ROW_INFO_VERSION_1, declared in the same include file, should be assigned to row_info.version by the subsystem. (Input)

value_ptr
is filled in by report_writer_ and points to the buffer where the row value is placed by the application subsystem's table manager procedure or the report_writer_$convert_and_move_row entrypoint. See "The Row Value Buffer" in Section 3 for additional information. (Output)

value_length
is filled in by report_writer_ and is the length of the buffer for the row value. This is set to the exact number of characters needed to contain the row value after conversion from internal data types to the non-varying, unaligned character data types report_writer_ uses. (Output)

number_of_columns
is filled in when the application subsystem allocates the row_info structure, by setting the variable row_info_init_number_of_columns to the number of columns that should be present in the table, before the allocate statement is executed. (Input)

current_column_number
is set to zero by report_writer_ and can be used by the application subsystem's table manager

procedure to place individual column values into the row value buffer. See "The Row Value Buffer" in Section 3 for additional information. (Output)

names
are filled in by the application subsystem, and are the names of the individual columns. These names must not contain whitespace because of the request line syntax of the report_writer_ standard requests. (Input)

descriptors
are filled in by the application subsystem. They are standard data type descriptors for the column data as it appears in the source file that the subsystem's table manager retrieves from. In many cases they describe columns that are not nonvarying, unaligned character data types, and report_writer_ must convert the column values to nonvarying character strings. It uses these data descriptors to do data conversion. See "Data Conversions" in Section 3 for additional information. It also uses these descriptors when generating default report column widths, column alignment modes, etc.

If a descriptor is completely zero, the column is assumed by report_writer_ to be a nonvarying, unaligned character string. In this case, the lengths field is used to determine the length in characters of the column. The descriptor is then set by report_writer_ to a nonvarying, unaligned character descriptor and its size portion is set to the value found in the lengths field.

Refer to the Programmer's Reference Manual, for a description of Multics standard data type descriptors. Refer to the include files "arg_descriptor.incl.pl1" and "std_descriptor_types.incl.pl1" for structures and named constants to set descriptors. (Input/Output)

lengths
are filled in by report_writer_, and each is the number of characters needed to contain the column value after conversion from any supported data type to nonvarying, unaligned character format. In the case where the descriptor is completely zero, lengths are set by the caller. The report_writer_ examines the value of this field to determine the length in characters for the column. See "The Row Value Buffer" in Section 3 for additional information. (Input/Output)

indexes
are filled in by report_writer_, and each is the index into the row value buffer for a particular column. See "The Row Value Buffer" in Section 3 for additional information. (Input)

There are also three variables defined in this include file to provide a convenient method to access the row_value buffer. A description of these variables follow.

row_value variable
provides a method to refer to the complete row value. Its declaration is:

```
declare row_value char(row_info.value_length)
        based (row_info.value_ptr);
```

A PL/I program to set the row value completely blank could use the PL/I statement:

```
row_value = " ";
```

row_value_as_an_array variable
provides a method to refer to any single character within the row value. Its declaration is:

```
declare row_value_as_an_array (row_info.value_length)
        char(1) based (row_info.value_ptr);
```

A PL/I program to set the 15th character within the row_value to a blank could use the PL/I statement:

```
row_value_as_an_array (15) = " ";
```

column_value variable
provides a method to refer to a complete column value. Its declaration is:

```
declare column_value
        char  (row_info.column.lengths
                (row_info.current_column_number))
        based (addr (row_value_as_an_array
                (row_info.column.indexes
                  (row_info.current_column_number))));
```

A PL/I program to set all of the columns to blanks could use the PL/I code fragment:

```
do row_info.current_column_number = 1 to row_info.number_of_columns;
    column_value = " ";
end;
```

Entry: destroy_invocation

This entry destroys a report_writer_ invocation.

USAGE

```
declare report_writer_$destroy_invocation entry (ptr);
call report_writer_$destroy_invocation (report_writer_info_ptr);
```

ARGUMENTS

report_writer_info_ptr
is the pointer to the report_writer_ info structure. If this pointer is null, the call is ignored. This pointer is set to null after the invocation is destroyed. (Input/Output)

Entry: set_report_writer_info_ptr

This entry is used by complex subsystems that require more than one report_writer_ invocation to be present in the subsystem invocation. It takes the caller provided report_writer_ info pointer and places it in an internal location associated with the sci_ptr. When the next report_writer_ standard request is invoked, it uses the new report_writer_ info pointer now associated with its sci_ptr parameter.

USAGE

```
declare report_writer_$set_report_writer_info_ptr entry (ptr, ptr,
        fixed bin(35), char(*) varying);
call report_writer_$set_report_writer_info_ptr
        (report_writer_info_ptr, sci_ptr, code, message);
```

ARGUMENTS

report_writer_info_ptr
: is the pointer to a report_writer_ info structure. This pointer becomes the report_writer_ info pointer that is passed to the report_writer_ standard requests. (Input)

sci_ptr
: is the pointer to the ssu_ info structure returned to the subsystem by the ssu_$create_invocation entrypoint. (Input)

code
: is a standard error code. If this code is non-zero, the current report_writer_ info pointer remains in effect. (Output)

message
: if code is non-zero this parameter contains the reason for the failure. Declaring this argument as "character (128) varying" provides an area large enough to contain any returned error message (Output)

NOTES

The most common type of subsystem using report_writer_ is one where there is only one report_writer_ invocation present in the one subsystem invocation. If a subsystem requires more than one report_writer_ invocation to be present in the one subsystem invocation, call the report_writer_$create_invocation as many times as required, saving the returned report_writer_ info pointer after each call. This entrypoint is then called with the desired report_writer_ info pointer before any of the report_writer_ standard requests are used. The call to this entrypoint ensures that the correct report_writer_ info pointer is used by the report_writer_ standard requests.

Entry: set_table_manager

This entry is used by complex subsystems that require more than one table_manager procedure to be used in a report_writer_ invocation. It takes the caller provided table_manager name and creates entry variables to the expected entrypoints. (See the "report_writer_$create_invocation entrypoint" for more information on the table_manager procedure.) After this entrypoint is called, the next invocation of the report_writer_ display request uses this new table_manager procedure.

USAGE

```
declare report_writer_$set_table_manager entry (ptr, char(*),
        fixed bin(35), char(*) varying);
call report_writer_$set_table_manager
        (report_writer_info_ptr, table_manager_name, code, message);
```

ARGUMENTS

report_writer_info_ptr
is the pointer to the report_writer_ info structure returned by the report_writer_$create_invocation entrypoint. (Input)

table_manager_name
is the name of the new table_manager procedure. (Input)

code
is a standard error code. If this code is non-zero, the current subsystem table_manager procedure remains in effect. (Output)

message
if code is non-zero, this parameter contains the reason for the failure. Declaring this argument as "character (128) varying" provides an area large enough to contain any returned error message. (Output)

NOTES

The most common type of subsystem using report_writer_ is one where there is only one table_manager procedure required for the one report_writer_ invocation. If a subsystem wishes to switch between different table_manager procedures within one report_writer_ invocation, this entrypoint should be called with the name of the new table_manager procedure. The call to this entrypoint results in the new table_manager procedure being called the next time the report_writer_ display request is used.

SECTION 6

TABLE MANAGER SUBROUTINE DESCRIPTION

The table_manager procedure is written by the application programmer, and is part of the application that uses report_writer_. The table_manager procedure is responsible for retrieving rows from the source data file (or files) that the application wishes to use in formatting the report.

At specific times during the retrieval operation, report_writer_ calls the table_manager procedure to accomplish certain tasks. At the beginning of the retrieval, report_writer_ calls the create_table entrypoint of the table_manager procedure to prepare for data retrieval and to retrieve the first data row. Each time another row is needed during the formatting operation, report_writer_ calls the get_row entrypoint of the table_manager procedure. After the retrieval is complete, report_writer_ calls the delete_table entrypoint of the table_manager procedure. These three entrypoints must be present in the table_manager procedure, or the report_writer_$create_invocation entrypoint refuses to create a report_writer_ invocation.

There is an optional entrypoint in the table_manager procedure which can be provided by applications that support user selected data retrieval. An example of this type of application is the LINUS subsystem. LINUS provides a query statement that allows users to choose the data of interest to them. When the report_writer_ save_format_options request is invoked with the -query control argument, report_writer_ tries to find this optional entrypoint in the applications table_manager procedure. If this entrypoint is not present, report_writer_ prints an error message stating that the operation is not supported. If the entrypoint is present, report_writer_ calls it to obtain the requests which the user typed to select the data currently being retrieved. These requests are saved, along with the report layout, in an exec_com segment. A subsequent restore_format_option request identifying the exec_com causes these requests to be executed. The execution of these requests results in the same data being selected and the same format options set, so that the report can be produced at a later time.

The following paragraphs describe the table_manager procedure entrypoints. Detail is provided on how the report_writer_ declares the entrypoints, and how the report_writer_ calls the entrypoints. The parameters passed to these entrypoints by report_writer_ are also explained.

Entry: table_manager$create_table

This entry is called by report_writer_ when the display request is invoked with the -new_retrieval control argument (Default). It performs any initialization required to perform its data retrieval function. It then retrieves the first row from its source data file, and places it in the row_value buffer.

USAGE

```
declare table_manager$create_table entry (ptr, fixed bin(35));
call table_manager$create_table (info_ptr, code);
```

ARGUMENTS

info_ptr
: is the pointer to the application's info structure that is passed to the ssu_$create_invocation entrypoint by the application when the ssu_ invocation is created. (Input)

code
: is a standard Multics error code. If this entry finds the source data file empty when the retrieval of the first row is attempted, it sets this code to rw_error_$no_data. If this entry executes successfully it sets this code to zero. If this entry encounters any other problem during execution, it sets the code to indicate what the source of the problem is. (Output)

NOTES

The application's info structure must contain the row_info_ptr passed to report_writer_$define_columns. This points to the row_info structure, which in turn points to row_value buffer. table_manager$create_table must move the first row of data into this row_value buffer, either by a PL/I assignment or substr pseudovariable, or by calling report_writer_$convert_and_move_row.

Entry: table_manager$delete_table

This entry is called by report_writer_ when the formatting of the report is completed. After the last data row is retrieved, it performs any termination steps required by the application.

USAGE

```
declare table_manager$delete_table entry (ptr, fixed bin(35));
call table_manager$delete_table (info_ptr, code);
```

ARGUMENTS

info_ptr
: is the pointer to the application's info structure that is passed to the

ssu_$create_invocation entrypoint by the application when the ssu_ invocation is created. (Input)

code
is a standard Multics error code. If this entry executes successfully it sets this code to zero. If this entry encounters any problem during execution, it sets the code to indicate what the source of the problem is. (Output)

Entry: table_manager$get_query

This entry is called by report_writer_ when the save_format_options request is invoked with the -query control argument. It returns to report_writer_ the requests needed to select the data currently being displayed, so that these requests can be saved along with the report layout. If the application does not support user requests to select data, this entrypoint should not be provided.

USAGE

```
declare table_manager$get_query entry (ptr, ptr,
   fixed bin(21), fixed bin(35));
call table_manager$get_query (info_ptr, query_segment_ptr,
   query_length, code);
```

ARGUMENTS

info_ptr
is the pointer to the application's info structure that is passed to the ssu_$create_invocation entrypoint by the application when the ssu_ invocation is created. (Input)

query_segment_ptr
is the pointer to a segment provided by report_writer_, where the data selection requests are placed by this entrypoint. (Input)

query_length
is the length in characters of the returned data selection requests. This entrypoint sets the length to let report_writer_ know how many characters are being returned. (Output)

code
is a standard Multics error code. If this entry executes successfully it sets this code to zero. If this entry encounters any problem during execution, it sets the code to indicate what the source of the problem is. (Output)

NOTES

The data selection requests are dependent on the application, and should contain whatever requests are necessary to select the data currently being displayed. These requests are executed from within a subsystem exec_com when the saved report layout is restored by the report_writer_ restore_format_options request. An example of data selection statments specific to the LINUS subsystem follows.

```
input_query -force -brief -terminal_input
select * from employee
.
translate_query
```

If any ampersands are found within the data selection requests that would be interpreted by exec_com when the report layout is restored, they are protected by save_format_options with double ampersands before being placed in the saved report layout file. In the case of a subsystem like LINUS, a portion of a select statement that looked like:

```
& name = "Smith"
```

would be saved as:

```
&& name = "Smith"
```

Entry: table_manager$get_row

This entry is called by report_writer_ when a row is needed during the formatting of the report. It retrieves a row from its source data file and places it in the report_writer_ row value buffer.

USAGE

```
declare table_manager$get_row entry (ptr, fixed bin(35));
call table_manager$get_row (info_ptr, code);
```

ARGUMENTS

info_ptr
: is the pointer to the application's info structure that is passed to the ssu_$create_invocation entrypoint by the application when the ssu_ invocation is created. (Input)

code
: is a standard Multics error code. If this entry executes successfully it sets this code to zero. If this entry encounters an end-of-file on the source data file, it sets this code to error_table_$end_of_info. If this entry encounters any other problem during execution, it sets the code to indicate what the source of the problem is. (Output)

NOTES

The application's info structure must contain the row_info_ptr passed to report_writer_$define_columns. This points to the row_info structure, which in turn points to row_value buffer. table_manager$get_row must move the first row of data into this row_value buffer, either by a PL/I assignment or substr pseudovariable, or by calling report_writer_$convert_and_move_row.

SECTION 7

DISPLAY EMPLOYEE PL/1 EXAMPLE

The sample user session in Section 2 was done using the PL/I program shown on the following pages. This PL/I program uses the report_writer_ and ssu_ subroutines to construct an application subsystem that includes the standard ssu_ and report_writer_ requests. Line numbers are included in the program listings for purposes of commentary.

MAIN PROCEDURE

```
1  display_employee: proc;

2          call initialize;
3          on cleanup call destroy_invocation;
4          message = create_invocation (code);
5          if code ^= 0
6          then call com_err_ (code, "display_employee", message);
7          else call ssu_$listen (sci_ptr, null, (0));

8          call destroy_invocation;

9          return;
```

lines 1-9

This is the main procedure that is invoked when a user at the terminal types display_employee. Line 2 calls an internal procedure to initialize the state of the program. Line 3 establishes a cleanup handler that calls destroy_invocation if the cleanup condition is signalled. Line 4 creates the display_employee invocation. Lines 5 and 6 check the error code and display the code and message if non-zero. Line 7 reads request lines from the terminal if the code is zero. Line 8 destroys the display_employee invocation.

create_table entry

```
10 create_table: entry (
11        based_info_ptr_parm, /* input: points to based_info structure */
12        code_parm);          /* output: success or failure */

13        based_info_ptr = based_info_ptr_parm;
14        call dsl_$retrieve (based_info.data_base_index,
15             "-range (e employee) -select e", based_employee, code_parm);
16        if code_parm = 0
17        then call report_writer_$convert_and_move_row (
18             based_info.report_writer_info_ptr, based_info.value_ptrs);
19        else if code_parm = mrds_error_$tuple_not_found
20             then code_parm = rw_error_$no_data;

21        return;
```

delete_table entry

```
22 delete_table: entry (
23        based_info_ptr_parm, /* input: points to based_info structure */
24        code_parm);          /* output: success or failure */

25        code_parm = 0;

26        return;
```

get_row entry

```
27 get_row: entry (
28        based_info_ptr_parm, /* input: points to based_info structure */
29        code_parm);          /* output: success or failure */

30        based_info_ptr = based_info_ptr_parm;
31        call dsl_$retrieve (based_info.data_base_index,
32             "-another", based_employee, code_parm);
33        if code_parm = 0
34        then call report_writer_$convert_and_move_row (
35             based_info.report_writer_info_ptr, based_info.value_ptrs);
36        else if code_parm = mrds_error_$tuple_not_found
37             then code_parm = error_table_$end_of_info;

38        return;
```

lines 10-38

These are the table manager entries called by report_writer_ to start and stop the data retrieval, and to get rows. When they are called they are given a new stack frame, so they use the based_info structure to access the first stack frame's automatic variables. Lines 10 to 21 create a new table, and are called when the report_writer_ display request's -new_retrieval control argument is used. Line 14 retrieves the first tuple from dsl_, and line 17 calls a report_writer_ entry to convert the row to characters and move it into the row_value buffer, if the retrieval was successful. If not successful, lines 19 and 20 return rw_error_$no_data if the relation was empty, or the dsl_ error code if it was anything else.

lines 22-26

This is the entrypoint called by report_writer_ after the completion of the report. It does not have any operations to perform, so it just sets the code to zero and returns.

lines 27-38

This is the entrypoint called by report_writer_ to retrieve a row of data from the source file and move it into the row value buffer. It retrieves the tuple from dsl_, and calls a report_writer_ entry to convert the row and move it into the row value buffer if the error code is zero. If the code is non-zero it replaces the mrds_error_$tuple_not_found code with error_table_$end_of_info, and returns any other dsl_ error code back to report_writer_.

INTERNAL PROCEDURES

create_invocation

```
39 create_invocation: proc (

40        code_parm)                    /* output: success or failure */
41        returns (char (256) varying); /* output: reason for the failure */

42 declare ci_loop fixed bin;
43 declare ci_message char (256) varying;
44 declare code_parm fixed bin (35) parm;

45        code_parm = 0;

46        call dsl_$open ("employee_data_base",
47             info.data_base_index, EXCLUSIVE_RETRIEVE, code_parm);
48        if code_parm ^= 0
49        then return ("Unable to open employee_data_base.");

50        call dsl_$get_attribute_list (info.data_base_index, "employee",
51             work_area_ptr, mrds_attribute_list_structure_version,
52             mrds_attribute_list_ptr, code_parm);
53        if code_parm ^= 0
54        then return ("Unable to get the list of attributes for employee.");

55        call ssu_$create_invocation ("display_employee", "1.0",
56             based_info_ptr, null, "", sci_ptr, code_parm);
57        if code_parm ^= 0
58        then return ("Unable to create the ssu_ invocation.");

59        call report_writer_$create_invocation ("", sci_ptr,
60             info.report_writer_info_ptr, code_parm, ci_message);
61        if code_parm ^= 0
62        then return (ci_message);

63        call ssu_$add_request_table (sci_ptr,
64             addr (ssu_request_tables_$standard_requests), 100000, code_parm);
65        if code_parm ^= 0
66        then return ("Unable to add the ssu_ standard requests.");
67        call ssu_$add_info_dir (sci_ptr, ssu_info_directories_$standard_requests,
68                  100000, code_parm);
69        if code_parm n = 0
70        then return ("Unable to add the ssu_ info directories.");

71        row_info_init_number_of_columns = mrds_attribute_list.num_attrs_in_view;
72        allocate row_info in (work_area) set (row_info_ptr);
73        row_info.version = ROW_INFO_VERSION_1;
74        do ci_loop = 1 to row_info_init_number_of_columns;
75             row_info.column (ci_loop).names
76                  = rtrim (mrds_attribute_list.attribute (ci_loop).model_name);
77             row_info.column (ci_loop).descriptors
```

```
78                  = mrds_attribute_list.attribute (ci_loop).user_data_type;
79        end;

80        call report_writer_$define_columns (info.report_writer_info_ptr,
81             row_info_ptr, code_parm, ci_message);
82        if code_parm ^= 0
83        then return (ci_message);

84        info.employee_ptr = addr (employee);
85        info.value_ptrs (1) = addr (employee.name);
86        info.value_ptrs (2) = addr (employee.job);
87        info.value_ptrs (3) = addr (employee.salary);
88        info.value_ptrs (4) = addr (employee.age);
89        info.value_ptrs (5) = addr (employee.sex);
90        info.value_ptrs (6) = addr (employee.family);
91        info.value_ptrs (7) = addr (employee.state);
92        info.value_ptrs (8) = addr (employee.city);

93        return ("");

94   end create_invocation;
```

---

lines 39–90

This is the internal procedure that creates a display_employee invocation. Lines 46 to 49 open the data base that is expected to be in the user's working directory, and pass back an error message and code if the data base cannot be opened. Lines 50 to 54 have dsl_ fill in the mrds_attribute_list structure, and pass back an error message and code if the attribute list cannot be obtained. Lines 55 to 58 create an ssu_ invocation and pass back an error message and code if there is a failure. Lines 59 to 62 create a report_writer_ invocation, and pass back an error message and code if there is a failure. Lines 63 to 70 add the standard ssu_ requests and info directory. Lines 71 to 79 allocate the row_info structure and fill in the version number, and names and descriptors arrays. Lines 80 to 83 call a report_writer_ entry to inform it that a set of columns have been selected, passing back an error message and code if there is a failure. Lines 84 to 92 fill in the value_ptrs array to locations in the employee structure that dsl_ retrieves the row into. This is necessary so that the create_table and get_row entrypoints reference the automatic employee structure in the main procedures stack frame. Line 93 returns a zero length error message indicating a normal execution.

destroy_invocation

```
95 destroy_invocation: proc;

96 declare di_code fixed bin (35);

97         if info.data_base_index ^= 0
98         then call dsl_$close (info.data_base_index, di_code);
99         call report_writer_$destroy_invocation (info.report_writer_info_ptr);
100        if mrds_attribute_list_ptr ^= null
101        then free mrds_attribute_list in (work_area);
102        if row_info_ptr ^= null
103        then free row_info in (work_area);
104        call ssu_$destroy_invocation (sci_ptr);

105        return;

106    end destroy_invocation;
```

---

lines 95-106

This is the procedure called to destroy a display_employee invocation. It closes the data base if open, destroys the report_writer_ invocation, frees the mrds_attribute_list structure if it was allocated, frees the row_info structure if it was allocated, and destroys the ssu_ invocation.

initialize

```
107 initialize: proc;

108       based_info_ptr = addr (info);
109       info.data_base_index = 0;
110       info.report_writer_info_ptr = null;
111       info.value_ptrs (*) = null;
112       sci_ptr = null;
113       mrds_attribute_list_ptr = null;
114       row_info_ptr = null;
115       work_area_ptr = get_system_free_area_ ();

116       return;

117  end initialize;
```

lines 107-117

This is the procedure called to initialize for a display_employee invocation. It sets the info pointer to point to its own info structure; sets other pointers and the data base index to initial values so the destroy_invocation entrypoint can execute correctly; and sets the work_area_ptr to the area used for allocations.

DECLARATIONS

```
118 declare EXCLUSIVE_RETRIEVE fixed bin(35)
            internal static options (constant) init(3);
119 declare addr builtin;
120 declare 1 based_employee like employee based (based_info.employee_ptr);
121 declare 1 based_info like info based (based_info_ptr);
122 declare based_info_ptr ptr;
123 declare based_info_ptr_parm ptr parm;
124 declare cleanup condition;
125 declare code fixed bin(35);
126 declare code_parm fixed bin(35) parm;
127 declare com_err_ entry() options(variable);
128 declare dsl_$close entry() options(variable);
129 declare dsl_$get_attribute_list entry (fixed bin(35),
            char(*), ptr, fixed bin, ptr, fixed bin(35));
130 declare dsl_$open entry() options(variable);
131 declare dsl_$retrieve entry() options(variable);
132 declare 1 employee aligned,
133           2 name char(10) unaligned,
134           2 job fixed decimal(2,0) unaligned,
135           2 salary fixed decimal(7,2) unaligned,
136           2 age fixed decimal(2,0) unaligned,
137           2 sex char(1) unaligned,
138           2 family char(1) unaligned,
139           2 state char(2) unaligned,
140           2 city char(13) unaligned;
141 declare error_table_$end_of_info fixed bin(35) ext static;
142 declare get_system_free_area_ entry() returns(ptr);
143 declare 1 info aligned,
144           2 data_base_index fixed bin(35),
145           2 report_writer_info_ptr ptr,
146           2 employee_ptr ptr,
147           2 value_ptrs(8) ptr;
148 declare message char(256) varying;
149 declare mrds_error_$tuple_not_found fixed bin(35) ext static;
150 declare null builtin;
151 declare report_writer_$convert_and_move_row entry (ptr, (*) ptr);
152 declare report_writer_$create_invocation entry (char(*),
            ptr, ptr, fixed bin(35), char(*) varying);
153 declare report_writer_$define_columns entry (ptr, ptr,
            fixed bin(35), char(*) varying);
154 declare report_writer_$destroy_invocation entry (ptr);
155 declare rtrim builtin;
156 declare rw_error_$no_data fixed bin(35) ext static;
157 declare ssu_info_directories_$standard_requests char(168) external;
158 declare sci_ptr ptr;
159 declare ssu_$add_info_dir entry (ptr, char(*), fixed bin, fixed bin(35));
160 declare ssu_$add_request_table entry (ptr, ptr, fixed bin, fixed bin(35));
161 declare ssu_$create_invocation entry (char(*), char(*),
            ptr, ptr, char(*), ptr, fixed bin(35));
162 declare ssu_$destroy_invocation entry (ptr);
163 declare ssu_$listen entry (ptr, ptr, fixed bin(35));
```

```
164 declare ssu_request_tables_$standard_requests bit(36) aligned external;
165 declare sys_info$max_seg_size fixed bin(35) ext static;
166 declare work_area area (sys_info$max_seg_size) based (work_area_ptr);
167 declare work_area_ptr ptr;

168 %include mrds_attribute_list;

169 %include rw_row_info;

170 end display_employee;
```

lines 118-170
These are the declarations for the display_employee procedure.

Even though the columns are re-ordered (line 4 above), the user must still set and list them in the selected order sequence. For example:

```
69 ! display_employee:  sfo -wid 8 -default;lsfo -wid 8
     -width city                "13"
```

Although city appears on the page first (i.e., left column in above example), the column is still column 8.

```
71 ! display_employee:  sfo -co 7 8;lsfo -co
72   -column_order              "state city name job salary age sex family"
```

Notice that all columns were not named in the -column_order request above (line 71) and that the system defaults all names (line 72). Future displays of the report will have the columns reordered to 7 8 1 2 3 4 5 6 until changed by the user.

INDEX

!
  see exclamation mark

;
  see semicolon

abbreviations
  MRW (Multics Report Writer)

basic operation 1-2

exclamation mark (!) 2-1

format options 1-2
  active options 1-3
  general column 1-2
    also see "user session"
  general report 1-2
    also see "user session"
  specific column 1-3
    also see "user session"

formatting
  full page 1-9

MRW
  see abbreviations

overview 1-1

report elements
  default 1-4
    alignment 1-5
    folding and width 1-4
    page layout and titles 1-4
    separators 1-4
  optional 1-5
    active requests 1-7
    column titles 1-7
    counts and subcounts 1-8
    editing 1-6
    embedded controls and hyphenation 1-9
    excluding columns 1-7
    grouping 1-8
    headers/footers 1-6
    ordering of columns 1-8
    outlining 1-8
    page breaks 1-7

requests (cont.)
    separators and delimiters 1-8
    totals and subtotals 1-8

report_writer_
  overview and tutorial 3-1
    creating an invocation 3-1
    data table retrieval 3-3
    destroying an invocation 3-11
    report formatting..p0 3-11
    report preparation 3-11
  subroutine description 5-2
    convert_and_move_row 5-2
    create_invocation 5-3
    define_columns 5-4
    destroy_invocation 5-7
    set_report_writer_info_ptr 5-7
    set_table_manager 5-8

requests
  column_value 4-2
  display 4-3
  display_builtins 4-8
  list_format_options 4-9
  restore_format_options 4-13
  save_format_options 4-13
  set_format_options 4-15

semicolon (;) 2-8

subroutine (report_writer_)
  see report_writer_

table_manager
  subroutine description 6-1

user session 2-1
  control argument abbreviation 2-3
  display_employee PL/I example 7-1
  general column options 2-30
  general report options-1 2-4
  general report options-2 2-16
  request abbreviations 2-1
  restoring a saved report 2-28
  saving a report and resetting options 2-27
  special editing of a report 2-24
  specific column options 2-6

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE: MULTICS REPORT WRITER REFERENCE MANUAL

ORDER NO. GB63-00

DATED JANUARY 1985

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

PLEASE FOLD AND TAPE—
NOTE: U. S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

FOLD ALONG LINE

FOLD ALONG LINE

CUT ALONG LINE

Honeywell

Together, we can find the answers.

Honeywell

Honeywell Information Systems
U.S.A.: 200 Smith St., MS 486, Waltham, MA 02154
Canada: 155 Gordon Baker Rd., Willowdale, ON M2H 3N7
U.K.: Great West Rd., Brentford, Middlesex TW8 9DH **Italy**: 32 Via Pirelli, 20124 Milano
Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F. **Japan**: 2-2 Kanda Jimbo-cho, Chiyoda-ku, Tokyo
Australia: 124 Walker St., North Sydney, N.S.W. 2060 **S.E. Asia**: Mandarin Plaza, Tsimshatsui East, H.K.

43151, 7.5C785, Printed in U.S.A.

GB63-00