

HONEYWELL

MULTICS LOGICAL
INQUIRY AND
UPDATE SYSTEM
REFERENCE
MANUAL

SOFTWARE

MULTICS LOGICAL INQUIRY AND UPDATE SYSTEM REFERENCE MANUAL

SUBJECT

Description of the Multics Logical Inquiry and Update System

SPECIAL INSTRUCTIONS

This manual supersedes AZ49, Revision 2, dated June 1980, and its addenda AZ49-02A dated July 1981 and AZ49-02B dated April 1983. Refer to the Preface for "Significant Changes."

The manual has been extensively revised and reorganized. Throughout the manual, change bars in the margins indicate technical additions and asterisks denote deletions. Section 4 and Section 6 are new and do not contain change bars.

SOFTWARE SUPPORTED

Multics Software Release 10.2

ORDER NUMBER

AZ49-03

December 1983

Honeywell

PREFACE

This manual describes a facility used to access Multics Relational Data Store data bases and to prepare data for report generation.

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: the New Users' Introduction to Multics - Part I and Part II, Order No. CH24 and CH25 respectively.

Throughout this manual, references are frequently made to other Multics manuals. For convenience, these references are as follows:

DOCUMENT	REFERRED TO IN TEXT AS
<u>Multics Relational Data Store Reference Manual</u> , Order No. AW53	MRDS Manual
<u>Multics PL/I Language Specification</u> , Order No. AG94	PL/I Manual
<u>Multics WORDPRO Reference Manual</u> , Order No. AZ98	WORDPRO Manual
<u>Multics Report Program Generator Reference Manual</u> , Order No. CC69	MRPG Manual
<u>Multics Programmers' Reference Manual</u> , Order No. AG91	REF Manual
<u>Multics Subroutines and I/O Modules</u> , Order No. AG93	Subroutines Manual

Significant Changes in this Addendum

Section 5 -- added Linus request names in the page header, added a new linus request (opened_database), substituted long names for all abbrevs in request examples, and added active request capabilities to the following linus requests:

```
assign_values
del_scope
list_scope
list_values
open
set_scope
```

The information and specifications in this document are subject to change without notice. Consult your Honeywell Marketing Representative for product or service availability.

New Section 7 -- relocated Exec Com Facility description from Section 5.
Appendix A -- added parameters defining generation of `linus_lila_tokens`.

CONTENTS

		Page
Section 1	Selection Language	1-1
	Syntax and Semantics of the Selection Language	1-7
Section 2	Built-in and Installation-Defined Functions	2-1
	Built-in Functions	2-1
	abs	2-1
	after	2-1
	avg	2-2
	before	2-2
	ceil	2-2
	concat	2-3
	count	2-3
	floor	2-3
	index	2-4
	max	2-4
	min	2-4
	mod	2-5
	reverse	2-5
	round	2-5
	search	2-6
	substr	2-6
	sum	2-6
	verify	2-7
	Writing Nonstandard Functions	2-7
Section 3	Data Base Creation	3-1
Section 4	Report Writer	4-1
	System Overview	4-1
	Basic Operation	4-1
	Formatting Options	4-1
	Requests	4-4
	Default Report Elements	4-5
	Page Layout and Titles	4-5
	Separators	4-5
	Folding and Width	4-5
	Alignment	4-6
	Optional Report Elements	4-6
	Editing	4-6
	Headers/Footers	4-7
	Column Titles	4-8
	Active Requests	4-8
	Page Breaks	4-8
	Excluding Columns	4-8
	Ordering Of Columns	4-8
	Grouping	4-9
	Outlining	4-9
	Totals and Subtotals	4-9
	Counts and Subcounts	4-9
	Separators and Delimiters	4-9
	Format Document Controls and Hyphenation	4-10
	Full Page Formatting	4-10
	User Session	4-13
	General Report Options-1	4-16
	Specific Column Options	4-18
	General Report Options-2	4-26

CONTENTS (cont)

	Page
Special Editing of a Report	4-33
Saving a Report and Resetting Options	4-34
Restoring a Saved Report	4-35
General Column Options	4-36
 Section 5	
Command Description	5-1
linux	5-2
linux Requests	5-4
.	5-8
?	5-8
abbrev, ab	5-9
answer	5-9
apply, ap	5-11
assign_values, av	5-12
close, c	5-14
column_value, clv	5-14
create_list, cls	5-15
declare, decl	5-16
define_temp_table, dtt	5-17
del_scope, ds	5-19
delete, dl	5-20
delete_temp_table, dltt	5-21
display, di	5-21
display_builtins, dib	5-26
do	5-26
exec_com, ec	5-29
execute, e	5-30
format_line, fl	5-31
help	5-33
if	5-34
input_query, iq	5-35
list_db, ldb	5-36
list_format_options, lsfo	5-38
list_help, lh	5-41
list_requests, lr	5-42
list_scope, ls	5-43
list_values, lv	5-44
ltrim	5-44
modify, m	5-45
open, o	5-47
opened_database	5-48
picture, pic	5-49
print, pr	5-50
print_query, pq	5-52
qedx, qx	5-52
quit, q	5-52
report, rpt	5-53
restore_format_options, rsfo	5-53
rtrim	5-54
save_format_options, svfo	5-55
save_query, sq	5-57
set_format_options, sfo	5-57
set_mode, sm	5-65
set_scope, ss	5-66
store, s	5-68
store_from_data_file, sdf	5-70
string	5-71
subsystem_name	5-71
subsystem_version	5-72
translate_query, tq	5-72
write, w	5-73
write_data_file, wdf	5-74
 Section 6	
Obsolete Linus Control Arguments/Requests	6-1

CONTENTS (cont)

	Page
linus	6-2
linus Requests	6-4
invoke, i	6-4
lila	6-5
Macro Facility	6-8
Section 7	
Exec Com Facility	7-1
exec_com Facility	7-1
Appendix	
Static Data Parameters	A-1

SECTION 1

SELECTION LANGUAGE

The Logical Inquiry and Update System (LINUS) is a powerful, yet "easy-to-use" facility for accessing centralized Multics Relational Data Store (MRDS) data bases. However, it is also possible for users to define private data bases and utilize LINUS to access and maintain them (refer to Section 3). LINUS provides a complete data base management capability including both retrieval and update operations. Data to be selected is specified via a selection language which is a high-level nonprocedural language capable of being understood and used by individuals who are not necessarily computer specialists.

Several of the LINUS requests (e.g., modify, delete, and print) operate on well-defined subsets of a data base. These data base subsets are selected via query statements. The user views the data base as a set of tables containing rows and columns of data. LINUS allows the selection algorithm to be specified as a series of table lookup operations, very similar to the way an individual manually scans a set of tables for information. For example, envision a telephone directory as being a table with three columns of information: name, address, and phone number. This table contains one row of information for each individual listed in the directory. Normally, to find the phone number for John C. Smith, the name column is scanned for the name "Smith John C", and the value is taken from the phone number column in the same row. In LINUS, this operation is described as:

```
select number
from phone_book
where name = "Smith John C"
```

Various features of LINUS are introduced via examples referencing a data base consisting of the following five tables that describe the operation of a department store:

emp					
<u>name</u>	emp_no	dept	mgr	sal	comm

sales		
<u>dept</u>	<u>item</u>	vol

supply		
<u>supplier</u>	<u>items</u>	vol

loc	
<u>dept</u>	floor

class	
<u>item</u>	type

The emp table contains a row of information on every employee, giving employee name, employee number, department, manager's employee number, salary, and commission for the last year. The sales table gives the volume of sales for every item within each department. The supply table provides the volume of each item supplied by every supplier. The loc table gives the floor on which every department is located, and the class table specifies the type of each item.

In each of the tables, the underscored words denote key columns. Every row in a table is uniquely identified by its values in the key columns. The LINUS user need not be concerned with the key column concept except when using the modify and define_temp_table requests discussed later.

The basic component of the selection language is the select-from-where block, which is used to select column values from one or more tables where rows of the tables satisfy certain conditions. It should be noted that the indentation of the following examples is for readability only, and is not required in actual usage. In fact, the entire query may be contained in one line.

The select clause and the from clause must always be specified in a select-from-where block. The where clause of a block may be omitted, in which case all rows are returned.

Example 1

List all departments from the emp table.

```
select dept
from emp
```

This could alternately be written as:

```
select dept from emp
```

A select clause may contain one or more column names, or may contain an asterisk (*) which indicates that all columns from qualifying rows are to be selected.

Example 2

List all information pertaining to every employee whose salary is greater than \$8,000.

```
select *
from emp
where sal > 8000
```

More complex conditions may be specified in the where clause, as shown in the remaining examples in this section. Specifically, a where clause may contain one or more terms. Each term consists of a column name or an arithmetic expression; followed by a relational operator; followed by a column name, arithmetic expression, or constant. Allowable relational operators are:

```
>    greater than
<    less than
<=   less than or equal to (or not greater than)
>=   greater than or equal to (or not less than)
=    equal to
^=   not equal to
```

Terms within the where clause must be separated by logical operators, and may be grouped using parentheses () to explicitly specify order of evaluation. Allowable logical operators are:

```
&    logical conjunction (and)
|    logical inclusive (or)
^    logical negation (not)
```

Character string constants within terms must be enclosed within quotes ". If a quote is to appear within a character string, a double quote must be specified.

Example 3

Find the names and salaries of employees in the toy department who work for Anderson, whose employee number is 1423.

```
select name sal
from emp
where dept = "Toy" & mgr = 1423
```

Example 4

Arithmetic expression may be contained in both the select clause and the where clause. All columns used in any given arithmetic expression must be defined over the same domain. Allowable operators in an arithmetic expression are:

```
+ addition
- subtraction
* multiplication
/ division
```

Find the names of employees who are either in the Admin department or whose sum of salary and commission exceeds \$10,000.

```
select name
from emp
where dept = "Admin" | sal + comm > 10000
```

It is possible to specify more complex table lookup operations by using a select-from-where block as the last component of a term in the where clause. This indicates that the comparison specified in the term is to be performed for every value selected by the inner block. All inner select-from-where blocks must be delimited by braces {}.

Example 5

Find all items sold by departments located on the second floor.

```
select item
from sales
where dept = {select dept
              from loc
              where floor = 2}
```

One can apply set functions to the results of a select-from-where block, as shown by the following examples. Allowable set functions are: min, max, count, avg, sum, and user-defined functions. User-defined functions are discussed in Section 2 (Writing Nonstandard Functions) and in Section 4 (Declare Request).

Example 6

Find the average salary of employees in the shoe department.

```
avg {select sal
     from emp
     where dept = "Shoe"}
```

Example 7

Find all employees whose salary is greater than that of any employee in the shoe department.

```
select name
from emp
where sal > max {select sal
                from emp
                where dept = "Shoe"}
```

A select clause can also contain an arithmetic expression as shown in the following example.

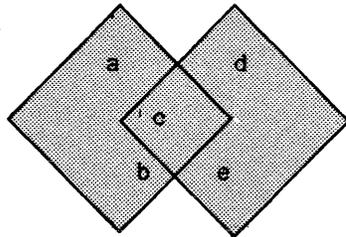
Example 8

Find each employee in the shoe department, together with her/his deviation from the average salary of that department.

```
select name sal - avg {select sal
                        from emp
                        where dept = "Shoe"}
from emp
where dept = "Shoe"
```

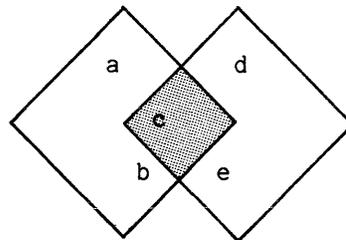
Set operations can be applied to the results of select-from-where blocks. In LILA the set operations are union, differ, and inter, which correspond to the union, difference, and intersection operations as normally defined. That is, the union of two sets consists of all items that belong to one or both of the sets. The intersection of two sets consists of those items belonging to both sets. The difference of two sets consists of those items which belong to the first set, but not to the second. For example, assume that set A contains the elements "a", "b", and "c" and the set B contains the elements "c", "d", and "e", then:

A union B (and B union A) is abcde (all items belong to one or both sets)



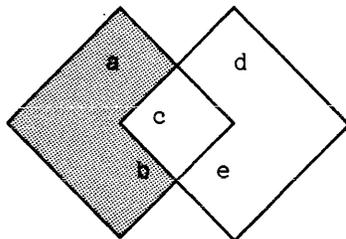
Set A Set B

A inter B (and B inter A) is c (items belong to both sets)



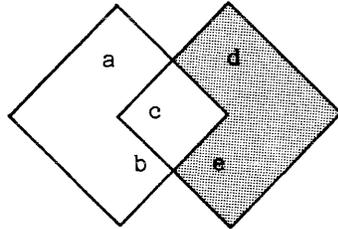
Set A Set B

A differ B is ab (items belong to the first, but not second set)



Set A Set B

B differ A is de (items belong to the first, but not second set)



Set A Set B

Example 9

Find those items which are supplied by Levi and sold in the men's department.

```
select item
from supply
where supplier = "Levi"
inter
select item
from sales
where dept = "Men"
```

Nesting of select-from-where blocks is possible in order to specify complex selection criteria.

Example 10

Find the total volume of type A items sold by departments on the second floor.

```
sum {select vol
      from sales
      where item = {select item
                   from class
                   where type = "A"} &
      dept = {select dept
              from loc
              where floor = 2}}
```

It is also acceptable to bypass the nested block notation and use table names to qualify column names (including *) within the select and where clauses. This qualification is accomplished by prefixing a column name with a table name followed by a dot (.). Whenever two or more table names are specified in the from clause of a block, all column names used within that block must be qualified. Using this approach, the above expression becomes:

```
sum {select sales.vol
      from sales class loc
      where sales.item = class.item & class.type = "A"
      & sales.dept = loc.dept & loc.floor = 2}
```

Finally, variables that assume rows of a designated table can be specified as values. In certain complex queries requiring comparisons among different rows of the same table, such row designators are required to resolve ambiguity. In essence, this allows a single table to be treated as multiple tables in the select and where clauses. A row designator is associated with a table by adding a prefix consisting of the row designator name followed by a colon (:) to the table name in the from clause. Several

row designators may be associated with a single table. The row designator is used in the select clause and where clause like a table name to qualify a column name.

Example 11

For all employees who earn more than their managers, select the employee's name and that of his manager.

```
select employee.name manager.name
from employee:emp manager:emp
where employee.mgr = manager.emp_no & employee.sal > manager.sal
```

These examples are intended as an introduction to basic features of the selection language. The information should allow the reader to write queries to satisfy a large class of data selection requirements. However, users should become familiar with the information in the remainder of this section for precise descriptions of the complete capabilities of the selection language.

SYNTAX AND SEMANTICS OF THE SELECTION LANGUAGE

A formal syntax is presented below using a metalanguage derived from Backus-Naur Form. The metalanguage symbols are defined as:

< > denotes a syntactical construct
::= means "is defined as"
[] denotes zero or one occurrence of (optional)
... denotes one or more occurrence of
| denotes the logical inclusive "OR"

The inclusion of an underscore character under any of the symbols distinguishes that symbol as not being a part of the metalanguage, but as being a part of the selection language syntax (see <bool_op> below).

```
<select_expr> ::=
    <set_value> | <select_set>

<set_value> ::=
    <set_fn> {<select_set>}

<set_fn> ::=
    <set_builtin> | <user_set_fn>

<select_set> ::=
    <select_block> | <select_set> <set_op> <select_block> |
    {<select_set>}

<set_op> ::=
    union | inter | differ

<select_block> ::=
    select <select_list> from <from_list>
    | select <select_list> from <from_list> where <conditional>

<select_list> ::=
    * | <select_item_list>
    | dup <select_item_list> | unique <select_item_list>
```

```

<select_item_list> ::=
    <select_item> | <select_item_list> <select_item>

<select_item> ::=
    <table_name>.* | <row_desig>.* | <expr>

<expr> ::=
    <column_spec> | <scalar_fn> (<arg_list>)
    | <expr> <arith_op> <arithmetic_constant>
    | <expr> <arith_op> <linus_variable>
    | <expr> <arith_op> <set_value>
    | <expr> <arith_op> <expr> | (<expr>)

<column_spec> ::=
    <column_name> | <table_name>.<column_name>
    | <row_desig>.<column_name>

<scalar_fn> ::=
    <scalar_builtin> | <user_scalar_fn>

<arg_list> ::=
    <arg> | <arg_list>, <arg>

<arg> ::=
    <expr> | <constant> | <set_value>

<arith_op> ::=
    + | - | * | /

<from_list> ::=
    <table_item> | <table_item> <from_list>

<table_item> ::=
    <table_name> | <row_tab_pair>

<row_tab_pair> ::=
    <row_desig>:<table_name>

<conditional> ::=
    <term> | <conditional> <bool_op> <term>
    | ^(<conditional>) | (<conditional>)

<term> ::=
    <expr> <rel_op> <atom>

<rel_op> ::=
    = | ^= | > | < | >= | <=

<bool_op> ::=
    & | |

<atom> ::=
    <expr> | <constant> | <set_value> | {<select_block>}

<constant> ::=
    <arithmetic_constant> | <bit_string_constant> |
    <character_string_constant> | <linus_variable>

<linus_variable> ::=
    !<identifier>

<table_name> ::=
    <identifier>

<row_desig> ::=
    <identifier>

```

```

<column_name> ::=
    <identifier>

<user_set_fn> ::=
    <fn_name>

<user_scalar_fn> ::=
    <fn_name>

<identifier> ::=
    <letter>[<letter>|<digit>|_|$]...

<letter> ::=
    A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|
    a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

<digit> ::=
    0|1|2|3|4|5|6|7|8|9

```

NOTES: A <set_builtin> is one of the built-in set functions described in Section 2. A <scalar_builtin> is one of the built-in scalar functions described in the same section. A <user_set_fn> and a <user_scalar_fn> must be declared according to the specifications contained in the declare request description (refer to Section 4).

If <select_set>s are within a <select_set>, they may optionally be grouped by braces {} to explicitly specify the order of evaluation. If not explicitly specified, intersections and differences are evaluated prior to unions, and evaluation proceeds from left to right for operators of equivalent precedence.

The <set_op>s union, inter, and differ correspond to the set operations union, intersection, and difference respectively.

The <select_list>s of all <select_block>s within a <select_set> must be union-compatible; that is, corresponding columns must take their values from the same domain. Also, such <select_list>s may not contain <expr>s other than <column_spec>s.

If the where clause is omitted from a <select_block>, all rows within the <from_list> qualify.

A <select_list> of * indicates that all column values from the row are to be selected. If the <select_list> is a *, then the <from_list> must be a <table_name>.

A specification of dup within a <select_list> indicates that duplicate sets of selected values are not to be eliminated, whereas a specification of unique indicates that duplicates are to be eliminated. If neither is specified, the default rule applies. The default is dup if a <set_fn> is to be applied to the selected values and is unique otherwise.

It should be noted that the use of scalar functions in the select clause may result in duplicate rows even though unique is specified. That is, LINUS applies scalar functions to column values returned from MRDS because of the select clause. MRDS actually does the duplicate elimination processing, but it does not know about built-in functions in the select clause.

A `<select_item>` of `<table_name>.*` or `<row_desig>.*` indicates that all columns from the row are to be selected. A `<table_name>` is the name of a previously defined temporary table, or a table defined within the data base. A `<row_desig>` is a row designator that is associated with a `<table_name>` in a `<from_list>`.

All `<column_spec>`s within an `<expr>` or `<arg_list>` must refer to column values from the same row.

Items within an `<expr>` may optionally be grouped by parentheses () to explicitly determine the order of evaluation. If not explicitly specified, all multiplications (*) and divisions (/) are performed before any additions (+) or subtractions (-). Multiplications and divisions are performed from left to right, as are additions and subtractions.

A `<row_tab_pair>` is used to specify the association of a row designator with a table. A `<row_desig>` must be unique for the entire `<select_block>`.

Items within a `<conditional>` may optionally be grouped by parentheses to explicitly specify the order of evaluation. If the order is not explicitly specified, the and (&) operators are evaluated prior to the or (|) operators; the and evaluation proceeds from left to right for operators of equivalent precedence.

The items `<arithmetic_constant>`, `<bit_string_constant>`, and `<character_string_constant>` are as defined in the PL/I Manual. An `<identifier>` is as defined in Multics PL/I with the exceptions that the dollar sign (\$) is not allowed and the hyphen (-) is allowed, so long as it is not the first or last character of the `<identifier>`. An `<fn_name>` is the same as the `<identifier>` except that the hyphen is not allowed.

SECTION 2

BUILT-IN AND INSTALLATION-DEFINED FUNCTIONS

BUILT-IN FUNCTIONS

The available built-in functions in LINUS are listed alphabetically and are immediately followed by a detailed description. Several of the built-in functions are used in the numbered examples included in Section 1.

abs	count	reverse
after	floor	round
avg	index	search
before	max	substr
ceil	min	sum
concat	mod	verify

Function: abs

This is an arithmetic scalar function whose reference has the form:

abs (X)

The result of this function is the absolute value of X, where X must be a numeric data item. X can only be real and the result value is a float decimal (59).

Function: after

This is a string scalar function whose reference has the form:

after (S1, S2)

The result is that portion of S1 that occurs to the right of the leftmost occurrence of S2 within S1. If S2 is a null string, the result is S1. If S2 does not occur within S1, the result is a null string. For example:

```
after ("abcde", "bc") = "de"
after ("abcde", "") = "abcde"
after ("abcde", "f") = ""
after ("10101"b, "10"b) = "101"b
```

Function: avg

This is an arithmetic set function whose reference has the form:

```
avg {select X
      from ...}
```

The result is the average (mean) of all X values selected. For example:

```
avg {select sal
      from emp
      where dept = "Shoe"}
```

is the average salary of all employees in the shoe department.

Function: before

This is a string scalar function whose reference has the form:

```
before (S1, S2)
```

The result is that portion of S1 that occurs to the left of the leftmost occurrence of S2 within S1. If S2 is a null string, the result is a null string. If S2 does not lie within S1, then the result is S1. For example:

```
before ("abcde", "bc") = "a"
before ("abcde", "") = ""
before ("abcde", "f") = "abcde"
before ("10101"b, "10"b) = ""b
```

Function: ceil

This is an arithmetic scalar function whose reference has the form:

```
ceil (X)
```

where X must be real. The result is the smallest integer (I) such that:

```
I >= X
```

For example:

```
ceil (20.5) = 21
ceil (-14.6) = -14
ceil (12) = 12
```

Function: concat

This is a string scalar function whose reference has the form:

```
concat (S1, S2)
```

The result is the concatenation of S1 and S2. For example:

```
concat ("abc", "de") = "abcde"  
concat ("101"b, "01"b) = "10101"b
```

Function: count

This is an arithmetic set function whose reference has the form:

```
count {select X1 X2 ...  
       from ...}
```

The result is the number of sets of Xi which are selected. For example:

```
count {select name  
       from emp  
       where dept = "Shoe"}
```

is the number of employees in the shoe department.

Function: floor

This is an arithmetic scalar function whose reference has the form:

```
floor (X)
```

where X is real. The result is the largest integer (I) such that:

```
I <= X
```

For example:

```
floor (20.5) = 20  
floor (-14.6) = -15  
floor (12) = 12
```

Function: index

This is a character string scalar function whose reference has the form:

```
index (S1, S2)
```

The result is an integer that is the position of the beginning of the leftmost occurrence of S2 within S1. If S2 is not in S1 then the result is 0. If S2 is a null string, the result is 0. For example:

```
index ("abcde", "bc") = 2  
index ("abcde", "f") = 0  
index ("abcde", "") = 0
```

Function: max

This is an arithmetic set function whose reference has the form:

```
max {select X  
      from ...}
```

The result is the largest X value selected. For example:

```
max {select sal  
      from emp  
      where dept = "Shoe"}
```

is the highest salary paid to any employee in the shoe department.

Function: min

This is an arithmetic set function whose reference has the form:

```
min {select X  
      from ...}
```

The result is the smallest X value selected. For example:

```
min {select sal  
      from emp  
      where dept = "Shoe"}
```

is the lowest salary paid to any employee in the shoe department.

Function: mod

This is an arithmetic scalar function whose reference has the form:

mod (X, Y)

where X and Y are real. The result is X modulus Y, such that:

if $Y \neq 0$ then $\text{mod}(X, Y) = X - Y * \text{floor}(X / Y)$
if $Y = 0$ then $\text{mod}(X, Y) = X$

For example:

mod (42, 5) = 2
mod (129.2867, 25) = 4.2867
mod (10, 0) = 10

Function: reverse

This is a string scalar function whose reference has the form:

reverse (S)

The result is a string which is the reverse of the value of S. For example:

reverse ("abcde") = "edcba"
reverse ("a") = "a"
reverse ("") = ""
reverse ("10110"b) = "01101"b

Function: round

This is an arithmetic scalar function whose reference has the form:

round (X, Q)

The result is a rounding of the value of X. When a value is rounded to n digits, the digits after the nth digit are dropped, and the nth digit is increased by 1 if the (n+1)th digit is 5 or greater for decimal, or 1 for binary. If X is float, then Q must be positive and the mantissa is rounded to Q digits. If X is fixed, it is rounded to a value that has Q fractional digits. For complex values, the function is defined by:

$\text{round}(X + Yi, Q) = \text{round}(X, Q) + \text{round}(Y, Q)i$

For negative values the following algorithm is used:

$\text{round}(x) = \text{round}(\text{abs}(X)) * -1$

For example:

round (183.629e6, 4) = 183.6e6
round (183.629, 2) = 183.63
round (183.629, -1) = 180
round (21.56 + 6.21i, 0) = 22 + 6i

Function: search

This is a character string scalar function whose reference has the form:

```
search (C1, C2)
```

The result is an integer value that is the position in C1 of the leftmost occurrence of any character contained in C2. If C1 does not contain any character in C2, the result is 0. For example:

```
search ("abcde", "b") = 2
search ("abcde", "") = 0
search ("abcde", "f") = 0
search ("abcde", "be") = 2
```

Function: substr

This is a string scalar function whose reference has the form:

```
substr (S, I, J)
```

-or-

```
substr (S, I)
```

The result is that portion of S that begins with the Ith character and has length J (if J is present), or is that portion of S that begins with the Ith character and continues to the end of S (if J is not present). For example:

```
substr ("abcde", 3, 2) = "cd"
substr ("abcde", 3, 0) = ""
substr ("abcde", 3) = "cde"
substr ("10101"b, 3) = "101"b
```

Function: sum

This is an arithmetic set function whose reference has the form:

```
sum {select X
      from ...}
```

The result is the total of all selected values. For example:

```
sum {select vol
      from sales
      where dept = "Shoe"}
```

provides the total sales volume of the shoe department.

Function: verify

This is a character string scalar function whose reference has the form:

```
verify (C1, C2)
```

The result is an integer value that is the position of the first character of C1 that does not occur in C2. When C1 contains only characters that are in C2, the result is 0. For example:

```
verify ("xyz", "abc") = 1
verify ("xyz", "xyz") = 0
verify ("abcde", "cba") = 4
```

WRITING NONSTANDARD FUNCTIONS

Nonstandard (or installation-defined) functions may be written in any language that accepts and processes a standard Multics argument list. It is assumed that these functions are written by experienced programmers. (Refer to the `linus` command "declare" request in Section 5 for an example of declaring a nonstandard function.)

Scalar functions are passed a complete standard Multics argument list containing argument pointers and descriptor pointers for both the input arguments and the return argument. The call is equivalent to:

```
return_val = fn_name$fn_name (in_arg1, ..., in_argn);
```

Set functions are called differently in that they are called several times and require three procedure entry points.

The first entry point is the `init` entry, which is called one time for evaluation of each set function. The method of evaluating a set function requires that data be accumulated in static storage. The purpose of this entry point is to initialize that static storage. The `init` entry is equivalent to:

```
call fn_name$fn_name_init;
```

The second entry point is the `calc` entry, which is called one time for each set of selected values. This entry is passed a complete standard Multics argument list containing argument pointers and descriptor pointers for all of the declared input arguments. The purpose of the `calc` entry point is to calculate (or accumulate) the value for the set function. The call to the `calc` entry is equivalent to:

```
call fn_name$fn_name_calc (in_arg1, ..., in_argn);
```

The third entry point of a set function is the `assign` entry. This entry is called after the `calc` entry has been called for all sets of selected values. The purpose of the `assign` entry is to actually assign a return value for the set function. The call to this entry is equivalent to:

```
return_val = fn_name$fn_name_assign ();
```

Two restrictions on arguments to nonstandard functions are:

1. No * extents are permitted.

2. Data types are restricted to those data types permitted in a MRDS data base. The use of pointers, entries, labels, structures, offsets, and arrays is not allowed.

Example of the pl1 source for a scalar function:

```
user_substr: proc(character_argument) returns(char(6));

dcl character_argument char(30);
dcl substr builtin;

    return(substr(character_argument, 1, 6));

end user_substr;
```

Example of pl1 source for a set function:

```
standard_deviation: proc;

dcl number_of_calls fixed binary internal static;
dcl (sum_of_x, sum_of_x_square) float decimal(59) internal static;
dcl (input_parameter, return_value) float decimal(59);

    return;    /* This entry point is only
                used for declaring the
                set function. */

standard_deviation_init: entry;
    /* Entry to initialize static data.
       This entry is used once before
       each evaluation of the set
       function. */

    number_of_calls = 0;
    sum_of_x = 0.0;
    sum_of_x_square = 0.0;
    return;

standard_deviation_calc: entry(parameter);
    /* Accumulate the needed
       information from the
       set of data. This entry
       point is called once for
       each row retrieved. */

    number_of_calls = number_of_calls + 1;
    sum_of_x = sum_of_x + parameter;
    sum_of_x_square = sum_of_x_square +
        (parameter ** 2);
    return;

standard_deviation_assign: entry returns(float decimal(59));
    /* This entry is called when
       there are no more rows
       to be looked at. It
       determines what the final
       value of the set function
       will be. */
```

```
if number_of_calls > 1 then
    return_value = ((number_of_calls *
                    sum_of_x_square -
                    sum_of_x** 2) /
                    (number_of_calls * (number_of_calls
                    - 1)))**.5;
else
    return_value = 0.0;
return(return_value);
end standard_deviation;
```

SECTION 3

DATA BASE CREATION

LINUS was designed primarily to allow users to access a centralized MRDS data base. However, it is also possible for users to define private data bases and utilize LINUS to access and maintain them. Users who wish to define a data base should refer to the MRDS Reference Manual, specifically to the introductory portion and to the description of the `create_mrds_db` command.

Certain differences in terminology between LINUS and MRDS are:

- A MRDS relation is a LINUS table.
- A MRDS tuple is a LINUS row.
- A MRDS attribute is a LINUS column.
- A MRDS domain is a LINUS domain and is the set of values that an attribute (column) may assume.

An example of the dept store data base, discussed in Section 1, may be created by invoking a text editor and creating the source segment, `dept_store.cmdb`:

```
domain:  name      char (30) unal,
         emp_no    fixed bin (17) unal,
         dept      char (12) unal,
         sal       fixed dec (13,2) unal,
         comm      fixed dec (13,2) unal,
         item      fixed bin (35) unal,
         vol       fixed bin (35) unal,
         supplier  char (30) unal,
         floor     fixed bin (8) unal,
         type      char (4) unal;

attribute:
         mgr      emp_no;

relation: emp (name* emp_no dept mgr sal comm),
          sales (dept* item* vol),
          supply (supplier* item* vol),
          loc (dept* floor),
          class (item* type);
```

Then the data base may be created by invoking the command:

```
create_mrds_db dept_store dept_store -list
```

The dept_store data base is now ready for loading, using the LINUS store request. (See linus command "store request" in Section 5.)

NOTE: LINUS can also open a data base from a submodel. (Refer to the create_mrds_dsm command in the MRDS manual for a description of submodels, and their creation.)

SECTION 4

REPORT WRITER

SYSTEM OVERVIEW

The LINUS report writer produces formatted reports from a relational data base. 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 of the report to the terminal, a file, or an io switch
- horizontal and vertical scrolling through the report

The report writer 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 report writer system retrieves rows of information (tuples) from a relational data base and produces a formatted output report. The rows retrieved are specified via a selection expression. (Refer to Section 1 for additional information and selection examples.)

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 as defined in the open model or submodel, 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 `linus` is first invoked, but can be changed by the user at any time. These values are retained for the entire `linus` 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 use of 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 5 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.

`format_line`

returns a single, quoted character string, formatted from an `ioa_control` string.

`list_format_options`

displays the names and values of formatting options.

`ltrim`

returns a character string trimmed of specified characters on the left.

`picture`

returns one or more values processed through a specified PL/I picture.

`restore_format_options`

restores saved report layouts.

`rtrim`

returns a character string trimmed of specified characters on the right.

`save_format_options`

saves current values of formatting options for future use.

`set_format_options`

changes/sets report formatting options.

`string`

returns a single character string formed by concatenating all of its arguments together, separated by single spaces.

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 (attribute name), which is found in the open submodel or model. If the column is the result of an expression or function invocation, the column title is "eN", where N begins at 1 and increases by 1 for each function invocation or expression encountered in the query. 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

Sometimes when formatting a report, the user finds that the report elements do not fit within the defined width. To rectify this situation, "folding" takes place. Folding can occur in two different ways. The first is "truncation." Truncation means that the value is truncated to the defined width and the last displayable character is replaced by the truncation character(s) (normally "**"). The second is "filling." 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 open model or submodel. The width chosen is the exact number of characters needed to contain the value after it is converted from the internal data base 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 in the open model or submodel. 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 `linus` active requests and Multics active functions. The `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 report writer 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 report writer during these two operations; the first time quote stripping occurs is when the `ssu $evaluate_active_string` subsystem utilities procedure is invoked. The `linus display_builtins active` request can be used to obtain built-ins like the current page number in a header/footer, and the `linus 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 linus picture active request can be used to provide editing features such as dollar signs and commas.

The user specifies linus 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 linus [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 in the query 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 in the query. This order can be changed by the user without having to go back and change the query.

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 work 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.

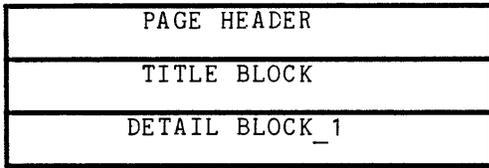
Format Document Controls and Hyphenation

The report writer 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

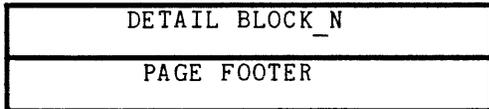
The report writer system formats a full page before any output is provided. It operates in this fashion because it is sometimes necessary to back up on a page and defer report elements to the next page so that associated report elements remain on the same page. A full page with all report elements present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

Formatted Page

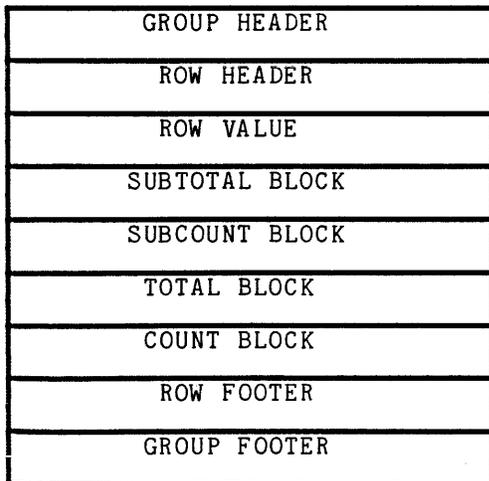


← expanded in diagram below

⋮



Detail Block



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

Backing up on a page is accomplished via a detection/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 works 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 works 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.

USER SESSION

The remainder of this section consists of report writer examples organized into a sample user session. 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 distinguish 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, blank lines have been removed in the examples for space consideration in this document. In most cases this can be recognized by the reader. For example:

```
55 ! linux: display -page 1
59 (system display)
```

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

Following is a list of request and control argument abbreviations used in the examples. They are included here for the purpose of saving the reader from referring to other sections if a term is unfamiliar.

REQUEST ABBREVIATIONS

clv	column value
di	display
dib	display_builtins
e	execute
ec	exec com
iq	input query
ldb	list_db
ls	list (Multics command level)
lsfo	list_format_options
o	open
pr	print (Multics command level)
q	quit
rsfo	restore_format_options
sfo	set_format_options
ss	set_scope
svfo	save_format_options
tq	translate_query

CONTROL ARGUMENT ABBREVIATIONS

-a	-all
-al	-alignment
-bf	-brief
-co	-column order
-dm	-delimiter
-ed	-editing

This page intentionally left blank.

-ex	-exclude
-fc	-force
-fold	-folding
-gft	-group_footer_trigger
-gfv	-group_footer_value
-ght	-group_header_trigger
-ghv	-group_header_value
-gr	-group
-it	-iteration
-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
-se	-selection_expression
-sep	-separator
-stt	-subtotal
-tc	-truncation -OR- -truncate
-td	-temp_dir
-ti	-terminal_input
-tl	-title_line
-tt	-total
-ttl	-title
-wid	-width

General Report Options-1

```
1 ! linux -it
2 ! linux: o employee r
3 ! linux: ss employee r u
4 ! linux: ldb -lg
```

TABLE	COLUMN	DECLARATION	DOMAIN	TYPE
employee (perm)				
	name	char (10)	name	key
	job	fixed dec (2) unal	job	key index
	salary	fixed dec (7,2) unal	salary	data index
	age	fixed dec (2) unal	age	data index
	sex	char (1)	sex	key index
	family	char (1)	family	data
	state	char (2)	state	data index
	city	char (13)	city	data index

```
5 ! linux: lsfo -a
  -delimiter                "!"
  -format_document_controls "off"
  -hyphenation               "off"
  -page_footer_value        ""
  -page_header_value        ""
  -page_length               "66"
  -page_width                "79"
  -title_line                "on"
  -truncation                "*"

```

```
6 ! linux: lsfo
  All of the formatting options are set to their default values.
  There are no column options defined.
```

```
7 ! linux: sfo -pw 0
8 ! linux: lsfo
  -page_width                "0"

```

```
9 ! linux: sfo -pw -default
10 ! linux: lsfo
  All of the formatting options are set to their default values.
  There are no column options defined.
```

line 1-4

Invoke `linus`, `open`, `set scope`, and `list` information about the data base.

line 5

List the names and values of "all" 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 `linus` session. For example, if the page width is changed, it remains at this new value until it is explicitly changed back, or until the `linus` session is terminated.

<code>-dm</code>	<code>"!"</code>	character used to delimit portions of header/footer.
<code>-fdc</code>	<code>"off"</code>	used when filling overlength character strings. If "off," ignore embedded controls.
<code>-hyphenation</code>	<code>"off"</code>	used when filling overlength character strings. If "off," do not attempt to hyphenate words.
<code>-pfv</code>	<code>""</code>	footer placed at bottom of each page.
<code>-phv</code>	<code>""</code>	header placed at top of each page.
<code>-pl</code>	<code>"66"</code>	length of each formatted page (number of lines).
<code>-pw</code>	<code>"79"</code>	width of each formatted page (number of character positions).
<code>-tl</code>	<code>"on"</code>	print the title line.
<code>-tc</code>	<code>"*"</code>	character that indicates truncation has occurred.

line 6-10

List options, set page width, list options again, reset page width, and list options once again. If line 10 included the `-all` control argument, the display would be the same as that following line 5.

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 query is processed.

```
1 ! linus: iq -bf
2 ! select * from employee
3 ! .
4 ! linus: tq
5 ! linus: 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 1-5
Invoke input_query, build query, translate query, and list the names and values of the column options.

line 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.

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

line 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)

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

line 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 name found in the open model or submodel)

line 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 ! linus: 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
Display page 1. Data is retrieved from the data base and formatted by default parameters.

line 59-118
System display

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

```
123 ! linus: sfo -wid state 5
125 ! linus: 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.
 - line 123-125
Set the width value for state column to "5" from its default value of "2" and display page 1.
 - line 129-188
System display -- note the difference in the state column header on line 129 from that displayed on line 59-61.

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

```
193 ! linus: sfo -wid 8 10
195 ! linus: 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)

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

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

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

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

```
270      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. Notice that multiple linus 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.

line 263-265

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

line 269-279

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

```
330 ! sfo -sep ** " | "
332 ! linus: 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
```

line 330-332

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

line 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 ! linus: sfo -al age left -ed salary -prompt

398 Enter -editing salary.

399 ! [pic \$zz,zz9v.99 [clv salary]]

400 ! .

402 ! linus: 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.

line 399-402

Edit request, termination, and display page 1.

line 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.


```

29 ! linus: sfo -phv -prompt
30 Enter -page header value.
31 ! ![e date]!Sample Report![e time]!
32 ! !!!!
33 ! .

```

```

35 ! linus: di -pg 1

```

```

39 04/29/83                               Sample Report                               10.26

```

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

line 31-35

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

line 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 ! linus: sfo -pw 0
62 ! linus: 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

line 60-62

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

line 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 ! linus: sfo -pfv -prompt
88 Enter -page_footer_value.
89 ! !!!!
90 ! !!- Page [dib page_number] -!!
91 ! .

```

```

93 ! linus: di -pg 1

```

```

97 04/29/83 Sample Report 10:26

```

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

line 89-93

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

line 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 ! linus: sfo -pl -default;di -pg 1

122 04/29/83

Sample Report

10:26

NAME	JOB	SALARY	AGE	S	F	STATE	CITY
				E <td>A<td></td><td></td></td>	A <td></td> <td></td>		
				X <td>M<td></td><td></td></td>	M <td></td> <td></td>		
					I <td></td> <td></td>		
					L <td></td> <td></td>		
					Y <td></td> <td></td>		
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 "default" (66 lines) from its previous setting of "26" (see line 1 of this example set).

line 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).

248 ! linus: 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 ! linus: 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 (line 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 !  linus:  sfo -wid sex 6 -ed sex "[ec sex_lookup [clv sex]]"
3 !  linus:  ..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.lec
12 ! q

```

```

14 ! linus:  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	norfolk

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.
 - line 3-12
Invoke the ted editor, append the following `exec_com` data (line 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
 - line 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).

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 ! linus: svfo EXAMPLE-1.fo.lec -query;sfo -rs;-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 linus subsystem exec_com (EXAMPLE-1.fo.lec) which can be restored later with the restore_format_options request. Then reset all options to their default values, and display page 1.

line 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 linus session by entering:

```
65 ! linus: 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:

```
1 ! linus
2 ! linus: o employer r
3 ! linus: ss employæ r u
4 ! linus: rsfo EXAMPLE-1.fo.1ec
5 ! linus: di -nr -pg 1
```

```
9 04/29/83 Sample Report 10:26
```

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

69

- Page 1 -

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

*

line 1-4

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?).

line 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.1ec) now resides in the user's working directory and may be printed 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 query. Every time a new query is performed, 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 ! linus: lsfo -co
2   -column_order           "name job salary age sex family state city"
4 ! linus: 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	FAMILY
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)							
norfolk	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.
 - line 8-67
System display -- note that the column order has been changed from that displayed in the previous example.

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

```
69 ! linus: 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 ! linus: 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.

```

74 ! linus: lsfo -ex
75   -exclude          ""

77 ! linus: 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.

line 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 ! linus: 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 ! linus: lsfo -gr
146   -group           ""

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

151 ! linus: 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	FAMILY
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)							
ca	fresno	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	m
		ordeman	1	\$15,200.01	21		d
. (6 data lines)							
ca	fresno	bane	13	\$15,200.01	50	female	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

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.

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 external to MRDS. The values must all be retrieved 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 ! linus: di -sort state city sex -kr -pg 2

220 04/29/83 Sample Report 10:34

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
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.

line 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 ! linus: di -kr -or -pg 2

The display results (provided by execution of line 281) would be an exact copy of that provided in line 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 ! linus: lsfo -out
      -outline                "sex"
```

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

```
292 04/29/83                               Sample Report                               10:36
```

STATE	CITY	NAME	JOB	SALARY	AGE	SEX	FAMILY
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)							
338	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	
ca	fresno	ordeman	1	\$15,200.01	21		d
		bane	13	\$15,200.01	50	female	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

```
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.
 - line 292-351
System display -- note 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.

This page intentionally left blank.

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 ! linus: di -or -kr -td [e wd] -pg 1
354 linus (display): Warning: The temp_dir >udd>Demo>linus_test 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 line 292-351 above, if it were included here.

```
356 ! linus: 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 line 292-351 above, if it were included here.

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

```
358 ! linus: .. ls
359 Segments = 224, Length = 353
360
361 rew    0  !BBBJNHFGnQJX1w.temp.0565
    .
    .
    .
369 r w    0  !BBBJNHFGmXFcFB.LINUS.table
370 r w    1  EXAMPLE-1.fo.1ec
371 r w    1  sex_lookup.1ec
    .
    .
    .
```

line 358

Escape out of `linus` and list the current contents of the working directory.

line 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 ! linus: lsfo -pb
376   -page_break      ""

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

Line 375 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).

04/29/83

Sample Report

10:39

STATE	CITY	NAME	JOB	SALARY	AGE	SEX	FAMILY
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

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

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	
	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	
	barker		14	\$32,800.01	78	male	d	
	g<MORE>		9	\$10,900.01	45		s	
	katz		5	\$12,400.01	70		m	
	m<MORE>		1	\$14,800.01	57		d	
	marcus		21	\$12,600.01	62		s	
	mccory		16	\$12,700.01	54		m	
	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			

This page intentionally left blank.

STATE	CITY	NAME	JOB	SALARY	AGE	SEX	FAMILIY	
ca	sacramento	meakin	3	\$12,900.01	71	male	m	
		monson	23	\$13,000.01	40		d	
		newman	19	\$ 4,200.01	68		s	
	san diego	payne	14	\$30,400.01	30		m	
		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
			gndt	13	\$14,700.01	60		s
			kelly	8	\$12,410.01	25		m
			macnabb	4	\$10,100.01	68		d
			markle	24	\$ 9,100.01	75		s
			m<MORE>	20	\$12,900.01	23		m
			media	15	\$12,300.01	45		d
			newman	10	\$11,300.01	62		s
	orrison		6	\$14,100.01	30		m	
	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	
	santa cruz		nguyen	21	\$12,700.01	53		d
		parce	16	\$12,900.01	68		s	
		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		

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, line numbers 1-8 are intentionally left blank.

```

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

14 ! linus: sfo -stt salary,state,reset

```

The subtotal inherits its width, editing request, etc. from the parent column. The width of the salary column must be increased or the subtotal will be 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 ! linus: lsfo -wid salary
17 -width salary "10"

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

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

29 ! linus: sfo -al salary decimal 12
31 ! linus: 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	FAMILY
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 rows of data)							
		paul	9	\$ 10,300.01	73		s
64	ak			\$ 262,056.19			

```

. (30 blank lines)

```

```

95 - Page 1 -

```

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

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

line 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.

line 35-281
System display -- note the inclusion of subtotals in the salary column (total by state -- see line 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 rows of data)						
		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								

This page intentionally left blank.

159 04/29/83

Sample Report

10:42

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
. (13 rows of data)							
	los angeles	aboe	4	\$ 10,201.01	41	female	m
		cowes	24	\$ 9,500.01	58		
. (17 rows of data)							
	sacramento	barrett	15	\$ 12,800.01	65	female	s
		gill	10	\$ 11,800.01	47		m
. (12 rows of data)							
		mccoy	7	\$ 14,000.01	77		s

219

221 04/29/83

Sample Report

10:43

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
. (16 rows of data)							
	san francisco	baur	18	\$ 10,100.01	79	female	d
		gnandt	13	\$ 14,700.01	60		s
. (17 rows of data)							
	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			\$ 1,302,223.95			
. (4 blank lines)							

281

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

```
1 ! linus: 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 ! linus: di -or -kr -pg 1,3
```

11 04/29/83 Sample Report 10:43

STATE	CITY	NAME	JOB	SALARY	AGE	SEX	FAMILY
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
		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

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)

- line 1 Request to set columns for subtotalling, with prompt.
- line 2 System display -- prompt.
- line 3-5 Set column values to be subtotaled, and terminate the prompt.

line 7
 Display pages 1 through 3.

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

10:43

73 04/29/83 Sample Report

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
	phoenix	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		
	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	
az	tucson	ordeman	1	\$ 15,200.01	21		d
			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	
		18	\$ 10,000.01	30	d			
		13	\$ 14,000.01	24	s			
		134	\$ 126,801.10	443	female			
			barker	14	\$ 32,800.01	78	male	d
			g<MORE>	9	\$ 10,900.01	45		s
			katz	5	\$ 12,400.01	70		m
		m<MORE>	1	\$ 14,800.01	57		d	
		marcus	21	\$ 12,600.01	62		s	
		mccory	16	\$ 12,700.01	54		m	
		mealey	11	\$ 21,600.01	36		d	
		nevitte	7	\$ 14,900.01	39		s	

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 ! linus: sfo -tt age salary job
199 ! linus: 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.

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		
	walla walla	colwell	18	\$ 10,200.01	38	female	m
		jenkins	13	\$ 15,001.01	41		d
		lawson	8	\$ 13,000.01	34		s
		m<MORE>	4	\$ 10,500.01	70		m
		mcclaus	24	\$ 9,000.01	51		d
		mcswain	20	\$ 12,400.01	35		s
		modlin	15	\$ 13,100.01	50		m
		nelson	10	\$ 11,700.01	37		d
		patrick	6	\$ 14,300.01	29		s
			118	\$ 109,201.09	385	female	

04/29/83

Sample Report

10:52

STATE	CITY	NAME	JOB	SALARY	AGE	SEX	FAMILIY		
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		
			condit	19	\$ 4,300.01		56	male	
			jochem	14	\$ 30,800.01		79		s
			lawter	9	\$ 10,400.01		74		m
			macleod	5	\$ 12,400.01		36		d
	m<MORE>	1	\$ 14,400.01	52		s			
	m<MORE>	21	\$ 12,400.01	60		m			
	mohin	16	\$ 12,000.01	34		d			
	nesline	11	\$ 20,500.01	37		s			
	p<MORE>	7	\$ 15,000.01	45		m			
	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		

04/29/83

Sample Report

10:53

STATE	CITY	NAME	JOB	SALARY	AGE	SEX	FAMILY
wi	racine	m<MORE>	2	\$ 12,600.01	35	female	m d s m d
		m<MORE>	22	\$ 18,700.01	38		
		moldt	18	\$ 10,200.01	67		
		n<MORE>	13	\$ 14,400.01	71		
		patton	8	\$ 12,100.01	27		
			114	\$ 118,310.09	407	female	
		baker	9	\$ 12,000.10	43	male	m d s m d s m d s
		gardner	5	\$ 12,300.01	29		
		jupiter	1	\$ 14,301.01	41		
		m<MORE>	21	\$ 12,600.01	32		
		mansour	16	\$ 13,100.01	46		
		mclung	11	\$ 21,100.01	39		
		modlin	7	\$ 14,300.01	60		
nelson	3	\$ 12,400.01	70				
o<MORE>	23	\$ 12,800.01	62				
	96	\$ 124,901.18	422	male			
			210	\$ 243,211.27	829		
			424	\$ 491,432.45	1744		
			12300	\$13,811,845.15	49106		

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

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

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

33 ! linus: 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
    Employee bambry is 66 years old and earns $11,501.01
    Employee justin is 78 years old and earns $12,000.01
    Employee cooke is 34 years old and earns $12,100.01
    Employee mcclenehan is 71 years old and earns $12,100.01
    Employee mead is 29 years old and earns $12,700.01
    Employee jones is 21 years old and earns $13,000.01
    Employee meadoows is 77 years old and earns $13,000.01
    Employee abel is 36 years old and earns $14,555.01
    Employee gaskins is 31 years old and earns $14,700.01
    Employee maclure is 53 years old and earns $14,700.01
    Employee mccormick is 67 years old and earns $14,900.01
    Employee macleod is 43 years old and earns $18,500.01
    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 -
```

line 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."

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

line 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 -".

line 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."

line 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 mclowsky 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 mccoys 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 meadows 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 coves 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
Employee mccullough is 62 years old and earns \$10,900.01
Employee gill is 47 years old and earns \$11,800.01
Employee macmannis is 54 years old and earns \$12,200.01
Employee newcomb is 36 years old and earns \$12,300.01
Employee kane is 24 years old and earns \$12,400.01
Employee barrett is 65 years old and earns \$12,800.01
Employee abodoura is 61 years old and earns \$12,900.01
Employee leonard is 25 years old and earns \$12,900.01
Employee meakin is 71 years old and earns \$12,900.01
Employee monson is 40 years old and earns \$13,000.01
Employee mccoys is 77 years old and earns \$14,000.01
Employee keene is 54 years old and earns \$14,100.01

225

- Page 3 -

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

This concludes the sample user session.

SECTION 5

COMMAND DESCRIPTION

This section contains a description of the `linus` command and its associated 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.

linus

linus

Name: linus

This command invokes linus to access an MRDS data base. It provides both retrieval and update operations. Data to be selected is specified via query statements.

Note: The linus command (pre-MR10.2 version) is described in Section 6.

Usage

linus {-control_args}

where control_args can be chosen from the following:

- abbrev, -ab
enables abbreviation expansion and editing of request lines.
- iteration, -it
recognizes parentheses in the request line to indicate request line iteration.
- no_abbrev, -nab
disables abbreviation expansion and editing of request lines. (Default)
- no_iteration, -nit
interprets parentheses in the request line literally (i.e., no iteration of request line). (Default)
- no_prompt, -npmt
turns off prompting of strings. This control argument can be overridden later (see set_mode request). (Default is prompt)
- no_start_up, -nsu
specifies that the subsystem start_up exec_com is not to be executed.
- profile path, -pf path
specifies the pathname of the profile used for abbreviation expansion. A profile suffix must be the last component to path; however, the suffix need not be supplied in the command line. This control argument implies -abbrev.
- prompt STR
sets the prompting string used by linus to STR. If STR contains embedded blanks, it must be enclosed in quotes. (Default linus prompt is "linus:".)
- request STR, -rq STR
executes STR as a linus request line before entering the request loop. This control argument cannot be used with the macro_path argument described in Section 6, and the requests specified by STR cannot contain the invoke request, also described in Section 6.
- start_up, -su
specifies that the subsystem start_up exec_com "start_up.lec" is

linus

linus

executed prior to entering the request loop. The start_up is searched for in the user's home directory, project directory, and then >site. (Default)

Notes

By default, linus prompts the user whenever input is expected (the string "linus:" is displayed at linus request level). Refer to the description of the set_mode request for information on how to turn off prompting.

Multics program_interrupt conditions are recognized and handled by linus. Thus, the user may interrupt any request and resume the linus session by invoking the Multics program_interrupt command. After the program_interrupt command, linus waits for the user to type further requests.

There is no data base creation facility within linus. Those users who wish to create their own data base should refer to Section 3 for information on the creation of an MRDS data base.

LINUS Requests

The following list summarizes all of the linus requests.

- identifies the linus subsystem, version number, and open data base.
- ? lists the available linus requests.
- abbrev, ab
turns abbreviation processing ON or OFF and changes profile segments.
- answer
supplies an answer to a question.
- apply, ap
places the current query in a temporary file, adds the file name to the supplied command line, and executes the Multics command line.
- assign_values, av
specifies that selected data is to be retrieved and that retrieved values are to be assigned to the designated variables.
- close, c
closes the currently open data base.
- column_value, clv
returns the value of the specified column for the current row, previous row, or next row.
- create_list, cls
specifies that selected data is to be retrieved and written to a Lister file to create a formatted report.
- declare, dcl
allows the user to declare user-written functions for later invocation within the selection expression.
- define_temp_table, dtt
specifies that selected data is to form a new temporary table, known only to the process, but which can be accessed by the process for retrieval in the same manner as data base tables.
- del_scope, ds
deletes all or a portion of the current scope of access in a shared data base.
- delete, dl
specifies that selected data is to be deleted from the data base.
- delete_temp_table, dltt
deletes the specified temporary table.
- 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.

do
substitutes args into the request_line and passes the result to the linus request processor.

exec_com, ec
executes the linus exec_com indicated by ec_path. The ec_path arguments are passed to the exec_com processor.

execute, e
executes a Multics command line after evaluating linus active requests.

format_line, fl
returns a single, quoted character string, formatted from an ioa_control string.

help
displays information about request names or topics. A list of available topics is produced by the list_help request.

if
conditionally executes a request.

input_query, iq
allows the entering of a query for data manipulation requests.

invoke, i (an OBSOLETE request moved to Section 6)

lila (an OBSOLETE request moved to Section 6)

list_db, ldb
lists specified information about the currently open data base.

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

list_help, lh
lists the available info segments whose names include a topic string.

list_requests, lr
lists information about linus requests.

list_scope, ls
lists the scope of access currently in force.

list_values, lv
lists the current value assigned to the designated linus variables.

ltrim
returns a character string trimmed of specified characters on the left.

modify, m
specifies that a selected portion of the data base is to be modified.

open, o
opens a specified data base, with either a data model or data submodel view, for linus processing.

opened_database, odb
returns "true" if there is an open data base and "false" if there is no open data base.

picture, pic
returns one or more values processed through a specified PL/I picture.

print, pr
specifies that selected data is to be retrieved and displayed on the terminal in default format.

print_query, pq
displays the current query.

qedx, qx
invokes the qedx editor with the current or a new query.

quit, q
terminates a linus session.

report, rpt
specifies that selected data is to be retrieved and used to create a formatted report via the Multics Report Program Generator (MRPG).

restore_format_options, rsfo
restores saved report layouts.

rtrim
returns a character string trimmed of specified characters on the right.

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

save_query, sq
saves the current query.

set_format_options, sfo
changes/sets report format options.

set_mode, sm
sets or resets modes for the current session.

set_scope, ss
defines the current scope of access within a shared data base (this, together with del_scope, provides concurrent usage control).

store, s
adds new rows to specified tables in the data base.

store_from_data_file, sdf
takes newrows from a file and adds them to the specified table in the data base.

string
returns a single character string formed by concatenating all of the strings together, separated by single spaces.

subsystem_name
displays the name of the subsystem, "linus".

subsystem_version
displays the current version of linus.

translate_query, tq
translates the current query, making it available for data manipulation requests.

linus

linus

write, w
specifies that selected data is to be retrieved and written to a file in the storage system. in the storage system.

write_data_file, wdf
retrieves selected data and writes it to a file in a format suitable for input to the store_from_data_file request.

The remainder of this section contains a detailed description of each request, including standard subsystem environmental requests (i.e., requests common to other subsystems such as abbrev, answer, do, etc). All examples show the prompting string "linus:" prior to lines of user input.

Request: .

This request identifies the linus subsystem, version number, and open data base.

Usage

Request: ?

This request displays the available linus requests.

Usage

?

Example

The following list is displayed when "?" is entered by the user in response to the linus prompt.

```

linus: ?
linus: Available linus requests:

.                help, h                report, rpt
?                if                restore_format_options,
abbrev, ab       input_query, iq          rsfo
answer          invoke, i                rtrim
apply, ap       lila                    save_format_options,
assign_values, av list_db, ldb            svfo
close, c        list_help, lh           save_query, sq
column_value, clv list_format_options,   set_format_options,
create_list, cls lsfo                    sfo
declare, dcl    list_requests, lr      set_mode, sm
define_temp_table, dtt list_scope, ls         set_scope, ss
delete_scope, ds list_values, lv        store, s
delete, dl      ltrim                   store_from_data_file,
delete_temp_table, dltt modify, m                sdf
display, di     open, o                 string
display_builtins, dib opened_database         subsystem_name
do              picture, pic            subsystem_version
exec_com, ec    print, pr               translate_query, tq
execute, e      print_query, pq        write, w
format_line, fl qedx, qx                write_date_file, wdf
quit, q

```

Type "list_requests" for a short description of the requests.

Request: abbrev, ab

This request controls abbreviation processing within the subsystem. As an active request, it returns "true" if abbreviation expansion of request lines is currently enabled within the subsystem and "false" otherwise.

Usage

```
ab {-control_args}
```

Usage as an Active Request

```
[ab]
```

where control_args can be chosen from the following (and cannot be used with the active request):

- off
specifies that abbreviations are not to be expanded.
- on
specifies that abbreviations should be expanded. (Default)
- profile path
specifies that the segment named by path is to be used as the profile segment; the profile suffix is added to path if not present. The segment named by path must exist.

Notes

This subsystem provides command line control arguments (-abbrev, -no_abbrev, -profile) to specify the initial state of abbreviation processing within the subsystem. For example, a Multics abbreviation can be defined to invoke the read_mail subsystem with a default profile as follows:

```
.ab rdm do "read_mail -abbrev -profile [hd]>mail_system &rf1"
```

If invoked with no arguments, this request enables abbreviation processing within the subsystem using the profile that was last used in this subsystem invocation. If abbreviation processing was not previously enabled, the profile in use at Multics command level is used; this profile is normally [home_dir]>Person_id.profile.

See the abbrev command in the Multics Commands for a description of abbreviation processing.

Request: answer

This request provides preset answers to questions asked by another request.

Usage

```
answer STR {-control_args} request_line
```

where:

1. STR
is the desired answer to any question. If the answer is more than one word, it must be enclosed in quotes. If STR is -query, the question is passed on to the user. The -query control argument is the only one that can be used in place of STR.
2. request_line
is any subsystem request line. It can contain any number of separate arguments (i.e., have spaces within it) and need not be enclosed in quotes.
3. control_args
can be chosen from the following:
 - brief, -bf
suppresses display (on user terminal) of both the question and the answer.
 - call STR
evaluates the active string STR to obtain the next answer in a sequence. The active string is constructed from subsystem active requests and Multics active strings (using the subsystem "execute" active request). The outermost level of brackets must be omitted (i.e., "forum_list -changed") and the entire string must be enclosed in quotes if it contains request processor special characters. The return value "true" is translated to "yes," and "false" to "no." All other return values are passed as is.
 - exclude STR, -ex STR
passes on, to the user or other handler, questions whose text matches STR. If STR is surrounded by slashes (/), it is interpreted as a qedx regular expression. Otherwise, answer tests whether STR is literally contained in the text of the question. Multiple occurrences of -match and -exclude are allowed (see "Notes" below). They apply to the entire request line.
 - match STR
answers only questions whose text matches STR. If STR is surrounded by slashes (/), it is interpreted as a qedx regular expression. Otherwise, answer tests whether STR is literally contained in the text of the question. Multiple occurrences of -match and -exclude are allowed (see "Notes" below). They apply to the entire request line.
 - query
skips the next answer in a sequence, passing the question on to the user. The answer is read from the user_i/o I/O switch.

- then STR
supplies the next answer in a sequence.
- times N
gives the previous answer (STR, -then STR, or -query) N times only (where N is an integer).

Notes

The answer request provides preset responses to questions by establishing an ON unit for the condition `command_question` and then executes the designated request line. If any request in the request line calls the `command_query_subroutine` (described in the Multics Subroutines) to ask a question, the ON unit is invoked to supply the answer. The ON unit is reverted when the answer request returns to subsystem request level. See "List of System Conditions and Default Handlers" in the REF Manual for a discussion of the `command_question` condition.

If a question is asked that requires a yes or no answer, and the preset answer is neither "yes" or "no," the ON unit is not invoked.

The last answer specified is issued as many times as necessary, unless followed by the `-times N` control argument.

The `-match` and `-exclude` control arguments are applied in the order specified. Each `-match` causes a given question to be answered if it matches STR; each `-exclude` causes it to be passed on if it matches STR. A question excluded by the `-exclude` control argument is reconsidered if it matches a `-match` later in the request line. For example, the request line:

```
answer yes -match /fortran/ -exclude /fortran_io/ -match /^fortran_io/
```

answers questions containing the string "fortran", except that it does not answer questions containing "fortran_io". It does, however, answer questions beginning with "fortran_io".

Request: apply, ap

This request places the current query into a temporary file, adds the pathname of the file to the end of the supplied command line, and executes the resulting Multics command line. If there is no current query, or the `-new` control argument is used, the created file is initially empty.

Usage

```
ap {-control_args} command_line
```

where:

1. `control_args`
can be chosen from the following:

- new specifies that an empty file be initially created.
- old specifies that the existing query be made available. (Default)

2. command_line
Is a Multics command line request.

Example

```
apply -new ted -pn
apply emacs
```

Request: assign_values, av

This request specifies that selected data is to be retrieved and the retrieved values assigned to designated linus variables. This capability allows information obtained from one retrieval to be used in subsequent data base accesses. A translated or translatable query must be available. As an active request, it returns "true" if data is successfully retrieved and "false" if the select statement fails.

Usage

```
av variable_list
```

Usage as an Active Request

```
[av variable_list]
```

where variable_list is a list of one or more variable names.

Notes

A variable name is an alphanumeric character string, from 1 to 32 characters in length, which must begin with an exclamation mark (!). The underscore (_) and hyphen (-) may also be included, but the exclamation mark cannot appear elsewhere in the name. The specification in an assign_values request is the only declaration required. If the same variable is specified in several assign_values requests, its value is reassigned in each of those requests. Variable names and values are preserved across data base openings and closings within the same linus session.

Variables specified in the assign_values request are unrelated to row designators in the query.

Retrieved data is assigned to variables in the variable_list in the order retrieved. Retrieval ceases when all selected data is exhausted or when all variables in the variable_list are exhausted, whichever occurs first. In the case of the retrieved data being exhausted before the variable_list, the following occurs: previously assigned variables that occur in a variable_list but are not assigned new values by this assign_value request retain their previous value. New variables in the variable_list that are not assigned values are not created.

Variable names are global within a linus session (i.e., like variable names occurring in different linus exec_coms refer to the same variable) if the exec_coms are used in the same linus session.

Example

List the employees whose total compensation is above the department store average, and then list those employees who are below the average.

```
linus: input_query -force
Query:
avg {select sal + comm from emp}
.
```

```
linus: assign_values !av_comp
linus: input_query -force
Query:
select name sal + comm
from emp
where sal + comm > !avg_comp
.
```

```
linus: print
```

```
name                f(emp)
Smith, John         10000
Jones, Al           12000
Johnson, Betty     11000
(END)
```

```
linus: qedx
3s/>/</
1,$ p
select name sal + comm
from emp
where sal + comm < !avg_comp
write
quit
```

```
linus: print
```

```
name                f(emp)
Anderson, Carol     8000
(END)
```

close

column_value

Request: close, c

This request closes the currently open data base.

Usage

c

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 as defined in the open model/submodel, or the number of the column in the query.
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. (If this control argument is not used the default value for STR is ".")
 - 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

Examples

```
[column_value foo]
[column_value 3]
[column_value foo -previous_row]
[column_value foo -next_row -default NULL]
```

Request: create_list, cls

This request specifies that selected data is to be retrieved and written to a specified Lister file. This file can be manipulated via Lister commands to create a formatted report. A translated or translatable query must be available. Refer to the WORDPRO Manual for a complete description of Lister.

Usage

```
cls path {-control_args}
```

where:

1. path
is the pathname of a Multics file into which the selected data is to be written. The data is written in a form suitable for processing by Lister. The suffix lister is appended to the pathname (if not present in the invocation) and the file is created if it does not exist. If the file currently exists, it is truncated unless the -extend control argument is specified.
2. control_args
can be chosen from the following:
 - extend
specifies that if the Lister file already exists, it is to be added to rather than truncated. The field names (either default or explicitly specified) must be identical to those defined in the existing file.
 - field_names STR, -fn STR
explicitly specifies the field names in the Lister file being created or extended. STR is a list of field names that must correspond in order and quantity to the items specified in the select clause of the associated query. This control argument must be specified if the query select clause contains an expression. If not specified, the names of the selected data base columns become the lister file field names.

NOTE: The values being written to the lister file have all leading and trailing blanks stripped off.

Examples

A query to create a Lister file containing the names and salaries for all employees in the Shoe department is:

```
linus: input_query -force
Query:
select name sal
from emp
where dept = "Shoe"
```

A request to create the Lister file "shoe_sal.lister" with the field names "name" and "sal" is:

```
linus: create_list shoe_sal
```

The same file could be created with the field names "name" and "salary" with the request:

```
linus: create_list shoe_sal -field_names name salary
```

Request: declare, dcl

This request allows the user to declare a nonstandard function which may be invoked in a subsequent query. A nonstandard function is any function not included in built-in functions listed in Section 2, and may be user-written or may be provided by the local installation. Two types of functions may be declared: set functions which operate on multiple sets of values (for example, sum {...}) and scalar functions which operate on one occurrence of a set of values.

Usage

```
dcl fn_name fn_type
```

where:

1. **fn_name**
is the name of the function being declared. The **fn_name** must be the name of an object segment that can be found using the search rules currently in effect.
2. **fn_type**
is the type of the function being declared. Two types are permitted, set or scalar. A set-type-function operates on multiple sets of selected values, whereas a scalar-type-function operates on one set of specified values. An example of a set function is:

```
avg {select salary
      from emp}
```

while a scalar function example would be:

```
substr (name 1, 5)
```

declare

define_temp_table

Notes

Scalar functions can accept column values as input from one table only, provided no row designators are used. If row designators are specified, column names must all be qualified with the same row designator.

Several built-in functions are provided as a standard part of linus. See Section 2 for a description of these functions. It is not necessary to declare built-in functions. If a declared function has the same name as a built-in function, the declared function, rather than the built-in function, is invoked when the function name is referenced.

Example

To find the average sales volume of all items made of cotton in a specific department, several assumptions are made: 1) that the item code contains encoded information indicating the material of which an item is made, 2) that the user-defined scalar function "material" returns this information, and 3) that there is a user-defined set function "dept_avg" that calculates the desired average, which is the total volume divided by the number of departments.

```
linus: declare material scalar
linus: declare dept_avg set
```

These functions may now be used in a query as:

```
linus: input_query -force
Query:
dept_avg {select dept vol
          from sales
          where material (item) = "cotton"}
```

Request: define_temp_table, dtt

This request causes selected data to be placed into a temporary table that can then be referenced as any other table in the data base for retrieval purposes. This feature is useful from an efficiency standpoint, since multiple retrievals of the same data can be avoided. A translated or translatable query must be available.

Usage

```
dtt table_name key_columns
```

where:

1. table_name
is the name of the temporary table. Subsequent references to this table must use this name. If a temporary table of this name already exists, it is redefined. This name may be from 1 to 32 characters long, must begin with an alphabetical character, and may be composed of alphanumeric characters plus the underscore (_) and the hyphen (-).

2. key_columns

are one or more column names specified in the associated select clause that become key columns in the temporary table. Key columns uniquely determine the rows of the temporary table; that is, the concatenation of the values of all key columns must be unique for each row of the temporary table. Duplicates are automatically eliminated.

Notes

The select clause of a query associated with a define_temp_table request cannot contain an expression. Only column names (qualified or unqualified, including *) are allowed.

All key columns must be explicitly specified in the associated select clause; that is, a key column cannot be one of those specified by a *. The order of the columns in the key of the temporary table is the order in which they appear in the select clause, not the order in the define_temp_table request.

Temporary tables cannot be updated, but can be accessed for retrieval only. Temporary tables that do not have any tuples may be created. Normally, a temporary table is created for the purpose of simplifying queries when data is to be selected from several tables in the data base.

Examples

If it is necessary to retrieve employee information from the department store data base depending upon the floor on which the employees are located, then a temporary table could be useful.

```
emp_loc (name, emp_no, mgr, sal, comm, floor)
```

The data for such a temporary table could be specified using the following query:

```
linus: input_query -force
Query:
select emp.name emp.emp_no emp.mgr emp.sal
       emp.comm loc.floor
from emp loc
where emp.dept = loc.dept
.
```

The table is then created with the request:

```
linus: define_temp_table emp_loc name
```

The query necessary to find the average salary of all employees located on the second floor would be:

```
linus: input_query -force
Query:
avg {select sal
      from emp_loc
      where floor = 2}
```

as opposed to the following, if the temporary table were not available:

```

linus: input_query -force
Query:
avg {select sal
      from emp
      where dept = {select dept
                   from loc
                   where floor = 2}}

```

Request: del_scope, ds

This request deletes all or a portion of the scope of access previously declared with a set_scope request, and is applicable only for shared (nonexclusive) opening modes. As an active request, it returns "true" if the scope is deleted and "false" if the delete scope fails.

Usage

```

ds table_name1 {permit_ops1 prevent_ops1 ... table_namen permit_opsn
               prevent_opsn}

```

Usage as an Active Request

```

[ds table-name1 {permit_ops1 prevent_ops1 ... table_namen
permit_opsn prevent_opsn}]

```

where:

1. table_name1
is the name of a nontemporary table within the data base for which all or a portion of the scope of access is to be deleted. If table_name1 is a *, then no additional arguments need be specified, and all of the user's current access scope is deleted, even if none is set.
2. permit_opsi
is a character string indicating which currently permitted operations are to be deleted from the access scope.
3. prevent_opsi
is a character string indicating which of the operations currently being prevented for other processes can be deleted from the access scope.

Note

The null operation is ignored for delete scope.

See the set_scope request for a definition of the operation codes and for a detailed discussion of the scope mechanism.

Examples

Do not change permission for the employee table but allow other processes to perform store, modify, and delete operations.

```
linus: del_scope emp n smd
```

Delete all of the current scope of access.

```
linus: del_scope *
```

Request: delete, dl

This request deletes selected rows from a single table within the data base. The data base must be open for update or exclusive_update and, if open for update, the affected table must be within the scope of access for delete. A translated or translatable query must be available.

Usage

dl

Note

The select clause of the associated query must specify columns from only one table and all columns from that table must be specified (use of * is recommended). The query must not contain any set operators (union, inter, or differ). The affected table cannot be a temporary table.

Example

Joe Smith has retired. Delete his employee record. The query would be:

```
Linus: input_query
Query:
select *
from emp
where name = "Joe Smith"
```

The deletion is then accomplished via the request:

```
linus: delete
```

Request: delete_temp_table, dltt

This request is used to delete temporary relations created for the current data base opening by the define_temp_table request.

Usage

dltt temp_table_name

where temp_table_name is a table name which has successfully been used in a define_temp_table request since the last successful open request.

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. A translated or translatable query must be available.

Usage

di {-control_args}

where control_args can be chosen from:

Note: The following list identifies all control arguments grouped by function. The argument descriptions are listed alphabetically, immediately after the function groupings.

CONTROLLING WARNING MESSAGES

-brief, -bf
-long, -lg

DISPLAYING PAGES AND PORTIONS OF PAGES

-all, -a
-character_positions, -chpsn
-page, -pg

DATA RETRIEVAL INITIATION AND TERMINATION

-discard_retrieval, -dsr
-keep_retrieval, -kr
-new_retrieval, -nr
-old_retrieval, -or

REPORT INITIATION AND TERMINATION

-discard_report, -dsrp
-keep_report, -krp
-new_report, -nrp
-old_report, -orp

SORTING RETRIEVED DATA

-sort

CONTROLLING REPORT OUTPUT

-extend
 -output_file, -of
 -output_switch, -osw
 -truncate, -tc

VIDEO SYSTEM SCROLLING FUNCTIONS

-enable_escape_keys, -eek
 -enable_function_keys, -efk
 -scroll
 -set_key, -sk
 -window, -win

MULTI-PASS REPORT FORMATTING

-passes, -pass

TEMPORARY STORAGE SPECIFICATION

-temp_dir, -td

- 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 for scrolling functions, rather than the function keys and arrow keys on the terminal. 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:

<u>Function Name</u>	<u>Key Sequence</u>
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:

<u>Function Name</u>	<u>Key Sequence</u>
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. (If this control argument is not provided, the default is to truncate an existing file.)

`-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.

`-long, -lg`
 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 from the data base. (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 (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, create 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 | corresponds 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 left arrow key |
| right_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 columns as defined in the open model/submodel, or numbers corresponding to the position of the columns in the selection expression. 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. (If the -extend control argument is not provided, the default is to truncate.)
- 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

```
display
display -output_file foo
display -keep_retrieval -sort bar -descending -non_case_sensitive
display -keep_retrieval -keep_report -of foo1 -character_positions 1 132
display -old_retrieval -old_report -of foo2 -character_positions 133 260
display -pages 1 3 12,19 58,$ -output_switch foo
display -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: do

This request expands a request line by substituting the supplied arguments into the line before execution. As an active request, it returns the expanded `request_string` rather than executing it.

Usage

```
do request_string {args}
```

or:

```
do -control_args
```

Usage as an Active Request:

```
[do "request_string" args]
```

where:

1. `request_string`
is a request line in quotes.
2. `args`
are character string arguments that replace parameters in `request_string`.
3. `control_args`
can be chosen from the following to set the mode of operation:
 - absentee
establishes an `any_other` handler that catches all conditions and aborts execution of the request line without aborting the process.
 - brief, -bf
specifies that the expanded request line not be printed before execution. (Default)
 - go
specifies that the expanded request line be passed on for execution. (Default)
 - interactive
specifies that the `any_other` handler not be established. (Default)
 - long, -lg
displays the expanded request line before execution.
 - nogo
specifies that the expanded request line not be passed on for execution.

List of Parameters

Any sequence beginning with & in the request line is expanded by the do request using the arguments given on the request line.

- &I
is replaced by argI. I must be a digit from 1 to 9.
- &(I)
is replaced by argI. I may be any value.
- &qI
is replaced by argI with any quotes in argI doubled. I must be a digit from 1 to 9.
- &q(I)
is replaced by argI with any quotes in argI doubled. I may be any value.
- &rI
is replaced by argI surrounded by level quotes with any contained quotes doubled. I must be a digit from 1 to 9.

&r(I)
is replaced by a requoted argI. I may be any value.

&fI
is replaced by all the arguments starting with argI. I must be a digit from 1 to 9.

&f(I)
is replaced by all the arguments starting with argI. I may be any value.

&qfI
is replaced by all the arguments starting with argI with any quotes doubled. I must be a digit from 1 to 9.

&qf(I)
is replaced by all the arguments starting with argI with quotes doubled. I may be any value.

&rI
is replaced by all the arguments starting with argI. Each argument is placed in level quotes with contained quotes doubled. I must be a digit from 1 to 9.

&rf(I)
is replaced by all the arguments starting with argI, requoted. I may be any value.

&&
is replaced by an ampersand.

&!
is replaced by a 15-character unique string. The string used is the same in every place where the &! appears in the request line.

&n
is replaced by the actual number of arguments supplied.

&f&n
is replaced by the last argument supplied.

Request: exec_com, ec

This request executes a program written in the exec_com language that is used to pass request lines to linus and to pass input lines to requests that read input. As an active request, it specifies a return value by use of the &return statement.

Usage

ec ec_path {ec_args}

Usage as an Active Request

[ec ec_path {ec_args}]

where:

1. `ec_path`
is the pathname of an `exec_com` program. An `lec` suffix is assumed if not specified.
2. `ec_args`
are optional arguments to the `exec_com` program and are substituted for parameter references in the program such as `&1`.

Notes

For a description of the `exec_com` language (both Version 1 and Version 2), type:

```
.. help v1ec v2ec
```

When evaluating a `linus exec_com` program, `linus` active requests are used rather than `Multics` active functions to evaluate the `&[...]` construct and the active string in an `&if` statement. The `execute` active request of `linus` can be used to evaluate `Multics` active strings within the `exec_com`. Refer to Section 7 for a description of how to write a `linus exec_com`.

Request: execute, e

This request executes the supplied line as a `Multics` command line. As an active request, it evaluates a `Multics` active string and returns the result to the subsystem request processor.

Usage

```
e STR
```

Usage as an Active Request

```
[e STR]
```

where `STR` is the `Multics` command line to be executed or the `Multics` active string to be evaluated. It need not be enclosed in quotes.

Notes

The recommended method to execute a `Multics` command line from within a subsystem is the `".."` escape sequence. The `execute` request is intended as a means of passing information from the subsystem to the `Multics` command processor.

All (), [], and "s in the given line are processed by the subsystem request processor and not the Multics command processor. This permits passing values of subsystem active requests to Multics commands when using the execute request, or passing values to Multics active functions for further manipulation before returning the values to the subsystem request processor for use within a request line.

Examples

The linus request line:

```
[execute max [column_value salary] [column_value commission]]
```

could be used as an editing request within a formatted report to return the largest value of the salary and commission columns.

The linus request line:

```
set format_options -page_header_value
[execute copy_characters - [list_format_options -page_width]]
```

could be used to set the page header to a line of hyphens which is the same width as the page width.

Request: format_line, fl

This request returns a single, quoted character string that is formatted from an ioa_control string and other optional arguments.

Usage

```
fl control_string {args}
```

Usage as an Active Request

```
[fl control_string {args}]
```

where:

1. control_string
is an ioa_control string used to format the return value of the active function. See "Notes" below.
2. args
are character strings substituted in the formatted return value, according to the ioa_control string.

Notes

The following `ioa_` control codes are allowed (refer to "`ioa_`" in the Subroutines Manual for additional detail):

<u>Control</u>	<u>Function</u>
<code>^a ^Na</code>	edit a character string in ASCII
<code>^d ^Nd</code>	edit a fixed-point number
<code>^e ^Ne</code>	edit a floating-point number in exponential form
<code>^f ^Nf ^N.Df ^.Df</code>	edit a floating-point number
<code>^i ^Ni</code>	edit a fixed-point number (same as <code>^d</code>)
<code>^o ^No</code>	edit a fixed-point number in octal
<code>^s ^Ns</code>	skip argument
<code>^[</code>	start an if/then/else or case select group
<code>^]</code>	limit the scope of a <code>^[</code>
<code>^(^N(</code>	start an iteration loop
<code>^)</code>	end an iteration loop
<code>^; ^N;</code>	used as a clause delimiter between <code>^[</code> and <code>^]</code>

In addition, any of the following carriage movement controls can be used:

```
^N/ ^N| ^N- ^Nx ^N^ ^R ^B
or
^/ ^| ^- ^x ^^
```

where `N` is an integer count or a "`v`". When "`v`" is given, an integer character string from the args is used for count. (For a complete description of these control strings see "`ioa_`" in the Subroutines Manual.)

If no optional arguments are given, the value returned depends on the specified `ioa_` control string.

Examples

In a formatted report the editing request:

```
[format_line "Height^-Weight^/^a^-^a" [column_value height] [column_value weight]]
```

might be expanded to return the string:

```
Height      Weight
6.1         175
```

The report editing request:

```
[format_line "^[Senior Citizen Discount^;Regular Discount^]"
 [execute ngreater [column_value age] 60 ]]
```

would be expanded to return the string:

```
Senior Citizen Discount
```

if the value of the age column was greater than 60.

Request: help

This request displays information about linux topics including detailed descriptions of linux requests.

Usage

```
help {topics} {-control_args}
```

where:

1. **topics**
specifies the topics on which information is to be displayed. The topics available within linux can be determined by using the `list_help` request.
2. **control_args**
can be chosen from the following:
 - brief, -bf
displays a summary of a request or active request, including the syntax, list of arguments, control arguments, etc.
 - search STRs, -srh STRs
displays the paragraph containing all the strings identified by STRs. (Default, the display begins at the top of the information.)
 - section STRs, -scn STRs
displays the section whose title contains all the strings identified by STRs. (Default, the display begins at the top of the information.)
 - title
displays section titles and section line counts, then asks if the user wants to see the first paragraph of information.

List of Responses

The most useful responses that can be given to questions asked by the help request are:

- displays "help" to identify the current interactive environment.
- .. **command_line**
treats the remainder of the response as a Multics command line.
- ?
displays a list of responses allowed.
- no, n
stops display of information and proceeds to the next topic, if any.
- quit, q
stops display of information and returns to subsystem request level.
- rest {-section}, r {-scn}
displays remaining information without intervening questions. If

-section is given, help displays the rest of the current section, without questions, and then asks if the user wants to see the next section.

search {STRs} {-top}, srh {STRs} {-t}
 skips to the next paragraph containing all the strings identified by STRs. If -top is given, searching starts at the top of the information. If STRs are omitted, help uses the STRs from the previous search response, or the -search control argument.

section {STRs} {-top}, scn {STRs} {-t}
 skips to the next section whose title contains all the strings identified by STRs. If -top is given, title searching starts at the top of the information. If STRs are omitted, help uses the STRs from the previous section response, or the -section control argument.

skip {-section}} {-seen}, s {-scn} {-seen}
 skips to the next paragraph. If -section is given, the request skips all paragraphs of the current section. If -seen is given, the request skips to the next paragraph that the user has not seen. Only one control argument is allowed in each skip response.

title {-top}
 displays titles and line counts of the sections that follow. If -top is given, help displays all section titles and repeats the previous question after titles are displayed.

yes, y
 prints the next paragraph of information on this topic.

Notes

If no topic names are given, the help request explains what help requests are available in the subsystem.

For a complete description of the control arguments and responses accepted by this request, type:

```
help help
```

Request: if

This request conditionally executes one of two request lines depending on the value of an active string. As an active request, it returns one of two character strings to the subsystem request processor depending on the value of an active string.

Usage

```
if expr -then line1 {-else line2}
```

Usage as an Active Request

```
[if expr -then STR1 {-else STR2}]
```

where:

1. `expr`
evaluates the active string as "true" or "false." The active string is constructed from subsystem active requests and Multics active strings (using the execute active request of the subsystem).
2. `line1`
executes the subsystem request line if `expr` is "true." If the request line contains any request processor characters, it must be enclosed in quotes.
3. `line2`
executes the subsystem request line if `expr` is "false." If omitted and `expr` is "false," no additional request line is executed. If the request line contains any request processor characters, it must be enclosed in quotes.
4. `STR1`
returns this value to the active request when `expr` is "true."
5. `STR2`
returns this value to the if active request when `expr` is "false." If omitted and the `expr` is "false," a null string is returned.

Request: input_query, iq

This request collects a query and makes it available for linux data manipulation requests.

Usage

```
iq {-control_args}
```

where control args can be chosen from the following:

- brief, -bf
specifies that the prompt "Query:" be suppressed when the query is entered from the terminal.
- force, -fc
specifies that the existing query be replaced. If a query exists and this control argument is not used, the user is asked if the existing query should be replaced. A negative response terminates the invocation of `input_query`.
- input_file path, -if path
specifies that the query be taken from the file named by path. If path does not contain the `lquery` suffix, it is assumed.

- long, -lg
specifies that the prompt "Query:" be displayed when the query is input from the terminal. (Default)
- no_force, -nfc
if a query exists, the user is asked if it should be replaced. (Default) A negative response terminates the invocation of input_query.
- terminal_input, -ti
specifies that the query be read from the terminal. (Default) A line consisting of only the single character "." terminates the input. Typing "\q" anywhere on a line also terminates the input. Typing "\f" anywhere on a line terminates the input and enters the user directly into the qedx editor with the query.

Example

```
input_query -if query_file -fc
input_query
Query:
select * from sales
.
```

Refer to Section 1 for examples of query statements.

Request: list_db, ldb

This request lists information about the data base that is currently open. Information which can be listed includes the pathname of the data base, the opening mode, table names, column names, and detailed information about each table and column. Information for both temporary and permanent tables is provided.

Usage

```
ldb {-control_args}
```

where control_args can be chosen from the following:

- long, -lg
specifies that all available information about columns is to be listed. This includes the name of the domain from which column values are derived and the PL/I-like declaration for this domain.
- names
specifies that only table and column names are to be listed.
- pathname, -pn
specifies that only the pathname of the data base, together with the opening mode, is to be listed.
- perm
specifies that information pertaining only to tables that are a permanent part of the data base is to be listed.

- table_names
specifies that only table names are to be listed.
- table STR, -tb STR
specifies that information pertaining only to tables named in STR is to be listed. STR is a list of permanent or temporary table names.
- temp
specifies that information pertaining only to temporary tables is to be listed.

Notes

If no control arguments are specified, -pathname and -table_names are assumed. If -table, -temp, or -perm is not specified, then information for all permanent and temporary tables is supplied.

If -table is specified, all other controls except -long and -pathname are ignored.

Only one of the following may be chosen:

- table_names
- names
- long

For non-DBA users of secure data bases (see the MRDS manual for the definitions of DBA and secured data bases), the following format changes will be seen because of possible security leaks:

1. Domain names will not be displayed.
2. Columns comprising the key of the table will not be shown as "key" columns. The first column in the key will be displayed as "index." All other columns will be shown as "data."

Examples

List the data base pathname and opening mode.

```
linus: list_db -pathname
>udd>Demo>dbmt>db>dept_store
update
```

List the names of all currently defined temporary tables.

```
linus: list_db -temp -table_names

TABLE

temp1
temp2
```

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:

Note: The following list identifies all format option arguments grouped by function. The argument descriptions are listed alphabetically, immediately after the function groupings.

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

- alignment column_id, -al column_id
displays the alignment mode within the display width for the specified column. (Also see "Notes".)
- column_order, -co
displays the order of columns in the detail line.
- count, -ct
displays the columns which have counts taken on them.
- delimiter, -dm
displays the character used to delimit the different portions of a header or footer.
- editing column_id, -ed column_id
displays the editing string for the specified column. (Also see "Notes".)
- exclude, -ex
displays the columns to be excluded in the detail line.
- folding column_id, -fold column_id
displays the folding action taken when the column value exceeds the display width for the specified column. (Also see "Notes".)
- 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).
- 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.
- hyphenation, -hph
displays hyphenation where possible for overlength values (on), or no hyphenation (off).
- outline, -cut
displays the columns which can duplicate suppression.
- page_break, -pb
displays the columns which can cause a break to a new page.
- 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.
- 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.
- 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. (Also see "Notes".)
- subcount, -set
displays the columns that have subcounts taken on them.
- subtotal, -stt
displays the columns that have subtotals taken on them.
- title column_id, -ttl column_id
displays the character string that is placed at the top of the page above the specified column. (Also see "Notes".)
- title_line, -tl
displays printing of the title line (on) or the suppression of the title line (off).
- total, -tt
displays the columns that have totals taken on them.
- truncation, -tc
displays the character or characters used to indicate truncation.
- width column_id, -wid column_id
displays the display width in the detail line for the specified column. (Also see "Notes".)

Notes

The variable `column_id` identifies the column name as defined in the open model/submodel, the number of the column in the query, or a star name which is matched against the column names.

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

```
list_format_options
list_format_options -all
list_format_options -width 1 -alignment salary
list_format_options -page_width -title ** -page_length
```

Request: list_help, lh

This request lists the names of all subsystem info segments pertaining to a given set of topics.

Usage

```
lh {topics}
```

where `topics` specifies the topics of interest. Any subsystem info segment that contains one of these topics as a substring is listed.

Notes

If no topics are given, all info segments available for the subsystem are displayed.

An info segment name is considered to match a topic only if that topic is at the beginning or end of a word within the segment name. Words in info segment names are bounded by the beginning and end of the segment name and by the characters period (.), hyphen (-), underscore (_), and dollar sign (\$). The info suffix is not considered when matching topics.

Examples

The request line:

```
list_help list
```

matches info segments named list_values, list_scope, list_db, etc., but would not match an info segment named prelisting, if such a segment existed.

Request: list_requests, lr

This request displays a brief description of selected subsystem requests.

Usage

```
lr {STRs} {-control_args}
```

where:

1. STRs specifies the requests to be displayed. Any request with a name containing one of these strings is displayed unless -exact is used, in which case the request name must match exactly one of these strings.
2. control_args can be chosen from the following:
 - all, -a includes undocumented and unimplemented requests in the display of requests eligible for matching the STR arguments.
 - exact displays only those requests whose names match exactly one of the STR arguments.

Notes

If no STRs are given, all requests are displayed.

A request name is considered to match a STR only if that STR is at the beginning or end of a word within the request name. Words in request names are bounded by the beginning and end of the request name and by the characters period (.), hyphen (-), underscore (_), and dollar sign (\$).

Examples

The request line:

```
list_requests values
```

matches requests named list_values and assign_values, but does not match a request named column_value.

Request: list_scope, ls

This request lists the current scope settings for permanent tables in the data base. As an active request, it returns the current scope settings.

Usage

ls {-control_arg}

Usage as an Active Request

[ls {-control_arg}]

where control_arg can be -table name-1 {... namen}, or -tb name-1 {... namen} which specifies that scope settings for only the named tables are to be listed. If -table is not specified, scope settings are listed for every permanent table in the data base that is in the current scope of access.

Examples

List the current scope of access.

linus: list_scope

Table	Permitted	Prevented
emp	rm	rsmd
sales	r	n

List the current scope of access for the sales and supply tables.

linus: list_scope -table sales supply

Table	Permitted	Prevented
sales	r	n
supply	n	n

Request: list_values, lv

This request lists the values of the designated linus variables. For information on creating linus variables see the assign_values request in this section. As an active request, it returns the value assigned to the designated linus variable.

Usage

```
lv {variable_1 ... variablen}
```

Usage as an Active Request

```
[lv variable]
```

where variable_i is one or more linus variable names (each name must begin with an exclamation point (!)). If this argument is omitted, then all existing linus variables are assumed to be the designated variables, and their values are displayed in the order that they were assigned their first values. Only one linus variable name can be supplied when used as an active request.

Example

```
linus: list_values abc
```

```
abc = 123
```

Request: ltrim

This request returns a character string trimmed of specified characters on the left.

Usage

```
ltrim STRa {STRb}
```

Usage as an Active Request

```
[ltrim STRa {STRb}]
```

Notes

The ltrim command or active function, finds the first character of STRa not in STRb, trims the characters from STRa preceding this character, and returns the trimmed result. Space characters are trimmed if STRb is omitted.

Examples

```
string [ltrim 000305.000 0]
305.000
```

```
string [ltrim " This is it. "]
This is it.
```

Request: modify, m

This request modifies selected data in the data base. The data base must be open for update or exclusive_update. If open for update, the table being updated must be within the current access scope for the modify operation. New values may be specified within the request line, or they may be entered interactively, in response to linus prompting. In both cases, the user is asked to verify the new values before the modification takes place, unless the -brief control argument is specified. A translated or translatable query must be available.

Usage

```
m {column_values} [-control_arg]
```

where:

1. column_values
are optional arguments and, if present, specify the new values that are to replace the current values of the data selected by the associated query. The column_values must be specified in the same order that the associated column names are listed in the select clause. If not present, linus requests the column_values individually by name.
2. control_arg
can be either -brief or -bf which specifies that verification of column_values is not to be done. If not present, linus displays a list of selected column names, together with the column_values as entered by the user, and requests that the user verify the correctness of the column_values before the modification operation proceeds. If the verification is negative, the modification does not take place. The user may reenter the modify request without again specifying the associated query.

Notes

New column values may be specified in two forms: 1) as constants or linus variables which have previously been set, or 2) as arithmetic expressions combining constants, linus variables, and possibly the name of the column being modified. All arithmetic expressions must be enclosed in parentheses. Any character string values entered via the request line and containing embedded white space must be enclosed in quotes.

The select clause of the associated query must specify columns from only one table, and only nonkey columns may be selected. The select clause associated with a

modify

modify

modify request may not contain arithmetic expressions, but is restricted to simple or qualified column names. Also, no set operators (union, inter, or differ) may appear in the query. The null character string ("") may be used only when modifying the data types:

```
character
character varying
bit varying
```

Examples

Give every employee a 10 percent raise. The query is:

```
linus: input_query
Query:
select sal
from emp
.
```

The modification is accomplished when iteration is on by:

```
linus: modify "(sal + .10 * sal)"

sal = (sal + .10 * sal)
OK?   yes
```

Al Jones has transferred to the shoe department. Update his employee record to indicate his new department and manager. The query is:

```
linus: input_query
Query:
select dept mgr
from emp
where name = "Al Jones"
.
```

The modification may be specified by:

```
linus: modify

dept?   Shoe
mgr?    1234

dept = Shoe
mgr = 1234
OK?    yes
```

Update the data base to indicate that the shoe department has moved to the third floor. The query is:

```
linus: input_query
Query:
select floor
from loc
where dept = "Shoe"
```

The modification may be specified by:

```
linus: modify 3 -brief
```

Request: open, o

This request opens a specified MRDS data base for accessing in the designated opening mode. The data base may be designated either by the pathname of the data base itself, or by the pathname of a data submodel associated with the data base. Only one data base may be open at any given time. As an active function, it returns "true" if the data base was successfully opened and "false" if it was not opened.

Usage

o path mode

Usage as an Active Request

[o path mode]

where:

1. path
is the pathname of an MRDS data base or of a data submodel associated with an MRDS data base. A data submodel is a user's view of the data base which may differ from the actual data base definition. See the MRDS Manual for a detailed discussion of data models and data submodels.
2. mode
is the usage mode for which the data base is to be opened. Modes can be specified either by their full names or by their abbreviations.
 - exclusive_retrieval, er
indicates that the user wishes only to retrieve data from the data base, but that concurrent access by other users for update is to be prohibited.
 - exclusive_update, eu
indicates that the user wishes to both retrieve and update information in the data base and that no concurrent access by other users is to be permitted.
 - retrieval, r
indicates that the user wishes only to retrieve data from the data base and allows concurrent access, for both update and retrieval, by other users. This mode requires that the user set scope for all tables to be touched (see the set_scope request).
 - update, u
indicates that the user wishes to both retrieve and update information in the data base and allows concurrent access, for both update and retrieval, by other users. This mode requires that the user set scope for all tables to be touched (see the set_scope request).

open

opened_database

Notes

For secure data bases, non-DBA users will be required to use the pathname of a secure data submodel. Refer to the MRDS manual for definitions of DBA, secure data submodel, and secure data base.

If the designated data base is already open by another user in a mode that conflicts with the mode designated in this open request, the open request is denied.

Several data bases may be opened and closed during a linux session. However, only one data base may be open at any given time.

Example

Open the department store data base for nonexclusive retrieval.

```
linux: open dept_store retrieval
```

Request: opened_database

This request (without the optional path), returns "true" if there is an open data base and "false" if there is no open data base. If path is provided, the request returns "true" if the specified mrds data base is currently open and "false" if the data base is not open. This request can only be used as an active request.

Usage as an Active Request

```
[oddb {path}]
```

where path is the pathname of a mrds data base or data submodel associated with a mrds data base.

Example

```
linux: string [opened_database]
false          /* there is no currently open data base */

linux: string [opened_database]
true           /* there is a currently open data base */

linux: string [opened_database foo]
false         /* the foo data base is not currently open */

linux: string [opened_database foo]
true          /* the foo data base is currently open */
```

picture

print

Request: picture, pic

This request returns one or more values processed through a specified PL/I picture.

Usage

```
pic pic_string values {-control_arg}
```

Usage as an Active Request

```
[pic pic_string values {-control_arg}]
```

where:

1. `pic_string`
is a valid PL/I picture as defined in the PL/I Reference Manual and the PL/I Language Specification.
2. `values`
are strings having data appropriate for editing into the picture. Each value must be convertible to the type implied by the picture specified. If multiple values are presented, the results are separated by single spaces. Any resulting value that contains a space is quoted.
3. `control_arg`
`-strip`
removes leading spaces from edited picture values; removes trailing zeros following a decimal point; removes a decimal point if it is the last character of a returned value.

Notes

For more information on PL/I picture and picture strings, see the PL/I Reference Manual, (Order NO. AM83) or the PL/I Language Specification (Order No. AG94)

Examples

The editing request in a formatted report:

```
[picture $99,999v.99 [column_value salary]]
```

returns the value \$27,922.41 if the value of the salary column was 27922.41.

Request: print, pr

This request specifies that selected data is to be retrieved and displayed on the user's terminal. The selected columns are displayed side-by-side with optional

column headers. The user may specify that a limit be placed on the number of rows to be displayed. A translated or translatable query must be available.

Usage

pr {-control_args}

where control args can be one or more of the following:

- all, -a
specifies that every row of information is to be displayed. The user is not queried.
- col_widths w1 ... wn, -cw w1 ... wn
explicitly specifies the width of each column to be displayed (in characters). If not present, the widths assume default values calculated from the data base definition of the items selected, or lengths of the column headers, whichever is larger. If this control argument is present, the specified widths must correspond in order and quantity to the items in the lila select clause. The column header is truncated if its length is greater than the column widths given. The wi may be integers or may be specified as "p.q", where p and q are precision and scale, respectively, of numeric data. Asterisks are printed if retrieved data cannot be printed in the column widths specified.
- col_widths_trunc, -cwt
is identical to the -cw control argument except that truncation occurs in cases where retrieved data contains more characters than the column widths specified. This argument is not compatible with -col_widths.
- max N
where N is a positive integer specifying that no more than N rows of information are to be displayed. If there are more than N rows, the user is queried as to whether more information is desired. Allowed responses to this query are: yes, to continue printing data and query after N more data lines are printed; no, to stop printing; or all, to print all remaining data without query. If -max N is not given, N is set to 10. This argument is incompatible with -all.
- no_end
specifies that the string "(END)" is to be suppressed when there is no more data to be printed.
- no_header, -nhe
specifies that column headers are not to be displayed. If not present, column headers consisting of column names are displayed if columns are selected. If an expression is selected, the column header is f(name), where name is the table or row designator name for the data base items appearing in the expression.

Notes

The columns are displayed side-by-side. The width of each column is determined from the data descriptions in the data base. Each column is separated from the next by two blanks. There is no pagination.

The current maximum total length for columns, two space separators, and trailing newline characters that make up the print line is 5000.

Example

Display the names of all employees in the shoe department, together with the sums of their salaries and commissions. The query is:

```
linus: input_query
Query:
select name sal + comm
from emp
where dept = "Shoe"
```

The retrieval is accomplished as follows:

```
linus: print

name          F(emp)
John Smith    10000
Al Jones      12000
Carol Anderson 8000
Betty Johnson 11000
(END)
```

Request: print_query, pq

This request prints (displays) or returns the current query.

Usage

pq

Usage as an Active Request

[pq]

Request: qedx, qx

This request invokes the qedx editor with the current query, or a new query. The edited query becomes the current query if the changes are saved before terminating qedx.

Usage

qx {-control_args}

where control_args can be chosen from the following:

- new specifies that qedx be given an empty buffer when invoked.
- old specifies that the existing query be made available for editing with qedx. (Default)

Note

The user must write (save) the changed query for it to become the current query.

Request: quit, q

This request terminates the linus session. If a data base is open at the time of this request, it is automatically closed.

Usage

q

Request: report, rpt

This request specifies that selected data is to be retrieved and used to generate a formatted report via the report I/O module and an existing Multics Report Program Generator (MRPG) object module. Refer to the MRPG Manual for a complete description of the MRPG facility. A translated or translatable query must be available when this request is specified.

Usage

rpt arg_string

where arg_string is a character string that must begin with the name of the MRPG object module, and must also contain any arguments required by the MRPG object module.

Note

The report is created by attaching the report file via report_ and opening it in stream_output mode. Each set of selected values is written as a line through report_. Within the MRPG program, the input from linus must be declared with the attribute, special.

Example

Create a formatted report containing the name, department, and salary of every employee. Assume that the MRPG object module, emp_report, creates the desired report. The query is:

```
linus: input_query
Query:
select name dept sal
from emp
.
```

The report is created by the following request:

```
linus: report emp_report
```

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 have a fo.lec suffix, one is assumed.

Notes

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

Examples

```
restore_format_options sample_display_format
restore_format_options another_display_format.fo.lec
```

rtrim

save_format_options

Request: rtrim

This request returns a character string trimmed of specified characters on the right.

Usage

rtrim STRa {STRb}

Usage as an Active Request

[rtrim STRa {STRb}]

Notes

The rtrim active function finds the last character of STRa not in STRb, trims the characters from STRa following this character, and returns the trimmed result. Space characters are trimmed if STRb is omitted.

Examples

```
string [rtrim 000305.000 0]
000305.
```

```
string [rtrim [ltrim 000305.000 0] 0]
305.
```

```
string X[rtrim " This is it. "]Y
X This is it.Y
```

Request: save_format_options, svfo

This request saves the current values of format options as a linux subsystem exec_com. The saved format can be restored with the restore_format_options request. The file is saved with a fo.lec suffix. Individual format options, active format options, or all of the format options can be saved. The query 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 a fo.lec 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) Type "help formatting_options.gi" 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.

save_format_options

set_format_options

-query

specifies that the current query is to be saved. A restore_format_options on the saved format also restores and makes the saved query current.

Examples

```
save_format_options report_layout
save_format_options report_layout -all
save_format_options report_layout -query
save_format_options report_layout -page_header_value -page_footer_value
save_format_options report_layout -page_header_value -width salary
save_format_options report_layout -width ** -page_footer_value
```

Request: save_query, sq

This request takes the current query and saves (writes) it to a file.

Usage

```
sq path
```

where path is the name of the saved file. If not present, a suffix of lquery is added to path.

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, linus sets the value of the format option to the system default. If -prompt is given for the value, linus 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:

Note: The following list identifies all format option arguments grouped by function. The argument descriptions are listed alphabetically, immediately after the function groupings.

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

-alignment column_id STR, -al column_id STR
column_id (see "Notes") 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 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.

-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.)

- 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 "!".)

- 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` (see "Notes") specifies which column the editing applies to. Multics active functions and linux active requests are normally used to provide additional editing. For example, the editing value:


```
[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.

- 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.)

- 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` (see "Notes") specifies which column the folding applies to. ~~STR set to "truncate", means the value of the column~~ is truncated to fit in the display width and the truncation character(s) is placed at the end of the value to indicate 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.)

- 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` 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.)

- 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.)
- 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.)
- 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.)

-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 66.)

-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 linus 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.)

-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.)

-separator column_id STR, -sep column_id STR

STR separates a column from the next one following it and column_id (see "Notes") 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>".)

-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.")

-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 (see "Notes") specifies which column the title applies to. (Default value of STR is the name of the column taken from the open model or submodel. In the case of expressions, the default value for STR is "eN", where N begins at 1 and is incremented by 1 for each additional expression found in the select list. 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.)

-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.)

-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.

-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 *.)

-width column_id N, -wid column_id N

N determines the display width for a column and column_id (see "Notes") 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 data type found in the data base, 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 linus 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 -- 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 linus 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.
- resets, -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

The variable `column_id` identifies the column name as defined in the open model/submodel, the number of the column in the query, or a star name which is used to match column names. If more than one table name is used in a select statement, the `column_name` is fully qualified (e.g., "table_name.column_name", or "row_designator.column_name"), otherwise the table name is unqualified (e.g., "table_name").

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:

```
set_format_options -page_width 80 -reset
```

is a valid request, but

```
set_format_options -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

```

set_format_options -width 1 25
set_format_options -title emp_name "Employee Name"
set_format_options -reset -page_width 80 -page_length 60
set_format_options -page_footer_value "!!-[display_builtins page_number]-!!"
set_format_options -page_header_value -prompt
Enter page_header_value.
![execute date]!LINUS REPORT![execute time]!
!!!!
..inl 0
!!--Page [display_builtins page_number]--!!
.

set_format_options -exclude exchange extension -width area_code 12
set_format_options -editing area_code "[format_line ^a/^a-^a [column_value area_code]
[column_value exchange] [column_value extension]]"

```

Request: set_mode, sm

This request sets or resets the specified mode and changes the prompt strings.

Usage

```
sm {mode1...modeN}
```

where `modei` may be one of the following:

```
iteration
    turns on request line iteration processing.
```

```
^iteration
    turns off request line iteration processing. (Default)
```

```
prompt
    turns on prompting. (Default)
```

```
^prompt
    turns off prompting.
```

```
set_linus_prompt_string STR, slups STR
    sets the linus prompt string to STR. If there are embedded blanks in STR,
    then STR must be enclosed in quotes. (Default is "linus:")
```

```
set_lila_prompt_string STR, slaps STR
    sets the lila prompt string to STR. If there are embedded blanks in STR,
    then STR must be enclosed in quotes. (Default is "->")
```

Note

The maximum prompt string length is 32 characters.

Example

Turn off prompting mode.

```
linus: set_mode ^prompt
```

Request: set_scope, ss

This request allows the user to define the current scope of access to the data base for nonexclusive opening modes. This request and the del_scope request are the means through which the user defines requirements to the linus concurrent access control mechanism. Every table that the user wishes to access for a given period must be included within the user's scope of access for that same period. As an active request, it returns "true" if the scope was set and "false" if the scope was not set.

For every table to be included in the current scope, the user specifies the types of access required, and also those types of access which are to be prohibited to other users. The scope of access is a dynamic entity, and may be varied to reflect the user's changing requirements during the life of a linus session. In order to prevent deadlock situations the current scope must be set to null with the del_scope request prior to issuing a set_scope request.

Usage

```
ss table_name1 permit_ops1 prevent_ops1 {... table_namen permit_opsn
  prevent_opsn} {-control_arg}
```

Usage as an Active Request

```
[ss table_name1 permit_ops1 prevent_ops1 {... table_namen permit_opsn
  prevent_ops_} {-control_arg}]
```

where:

1. table_name_i
is the name of a nontemporary table within the data base that is to be included in the current scope of access.
2. permit_ops_i
is a character string indicating which types of data base operations are to be permitted the user who is setting scope for the corresponding table. The character string is the concatenation of the codes for all operations to be permitted. See "Notes" for a description of the operation codes.
3. prevent_ops_i
is a character string similar to that for permit_ops_i indicating which

types of data base operations are to be denied other users for the corresponding table.

4. control_arg
 may be either -time seconds, or -tm seconds where seconds is an integer that specifies the wait time in seconds to be allowed for the requested scope to be granted. The default wait time is 30 seconds. If the requested scope cannot be granted within the specified wait time, the user is notified of the denial and may try again or may take other appropriate action.

Notes

Codes for operation types to be permitted or prevented are:

<u>Code</u>	<u>Operation</u>
d	delete
m	modify
n	null
r	retrieve
s	store
u	update (store, modify, delete)

It is recommended that users declare the minimum access scope necessary for any given operation and that the scope be maintained for only as long as it is needed. Declaration of unnecessarily large scopes is discouraged, as other users may be needlessly locked out of the data base.

The set_scope request is denied if the user currently has a nonnull scope in force. Therefore, all of the user's access scope must be deleted with a del_scope request prior to issuing a set_scope. The set_scope request must then specify the entire scope of access required by the user for a block of operations. This is in contrast to the del_scope request, where portions of the current scope may be deleted. If another user has a conflicting scope in force, the set_scope request is denied.

Specification of a modify, store, or delete permit_op causes a retrieve permit_op for the same table to be automatically requested.

The null (n) scope operation code is ignored, unless given by itself.

Example

Jim Jones, the manager of the shoe department, has retired and is being replaced by Al Smith. Update the employee table to reflect these changes, while ensuring that no other users access inconsistent data. This may be done in two steps.

Step One: Do the necessary retrieves

```
linus: input_query
Query:
select emp_no from emp
where name = "Al Smith"
.

linus: set_scope emp r n
linus: assign_values !smith_no
linus: qedx
```

```

2s/Al Smith/Jim Jones/
1,$ p
select emp_no from emp
where name = "Jim Jones"
write
quit

```

```
linus: assign_values !jones_no
```

Step Two: Modify and delete

```

linus: qedx
1s/emp_no/mgr/
2s/name = "Jim Jones"/mgr = !jones_no
1,$p
select mgr from emp
where mgr = !jones_no
write
quit

```

```

linus: del_scope *
linus: set_scope emp dm rdms
linus: modify !smith_no -brief
linus: qedx
1s/mgr/*/
2s/mgr/emp_no/
1,$ p
select * from emp
where emp_no = !jones_no
write
quit

```

```

linus: delete
linus: del_scope *

```

Notice that the only time it is necessary to prevent access to other users is while modify and delete are being accomplished.

Request: store, s

This request adds newrows to a designated table in the data base. The data base must be open for update or exclusive_update. If open for update, the table being stored must be within the current access scope for the store operation. Values being stored may be specified in one of three ways: 1) directly within the request line, 2) interactively in response to linus prompting, or 3) by placing the values in a Multics file and supplying the pathname as a control argument in the store request line. Using the first two methods, only a single row may be stored with one store request, whereas the third method (file input) allows the storing of multiple rows. Moreover, if the new row is being entered from the terminal (as opposed to file input), the user has the option of verifying the values prior to their being stored into the data base. (Also see the store_from_data_file request.)

Usage

```
s table_name {column_values} {-control_args}
```

where:

1. `table_name`
is the name of the table to which rows are being added. This must be the name of a nontemporary table.
2. `column_values`
are optional arguments and, if present, specify the column values comprising the new row being added. The `column_values` must be specified in the same order that the corresponding columns appear in the data base or the data submodel, whichever is applicable. Also, exactly one value must be specified for every column defined in the data base or data submodel.
3. `control_args`
can be one or more of the following:
 - brief, -bf
specifies that verification of `column_values` is not to be done. If not present, and if the `-input_file` control argument is not present, `linus` displays a list of column names, together with the `column_values` entered by the user, and requests that the user verify the correctness of the `column_values` before the store operation proceeds. If the verification is negative, the store does not take place, and the user must reenter the store request.
 - column_delimiter CHAR, -cdm CHAR
specifies that each `column_value`, in the file specified via `-input_file`, is separated from the next by the CHAR character. This control argument has meaning only if specified together with `-input_file`. If not present, each `column_value` is assumed to be delimited by one or more blanks.
 - input_file path, -if path
specifies that the `column_values` are to be taken from the Multics file designated by path. path is the pathname designating a Multics file suitable for processing by `vfile` in the stream_input opening mode. See "Notes" for a detailed description of the input file.
 - row_delimiter CHAR, -rdm CHAR
specifies that each row value is separated from the next by the CHAR character. If not present, each row value is assumed to be delimited by a newline (NL) character.

Notes

If `column_values` are not present in the request line and `-input_file` is not specified, then `linus` requests each `column_value` individually by name.

If `-input_file` is specified, the input file may contain `column_values` for more than one row. The input for each row is terminated by a newline character or the row delimiter character, if specified. In all cases, `column_values` are separated by blanks unless another delimiter is specified via `-column_delimiter`.

Examples

Add a new supplier to the supply table.

```
linus: store supply Acme 10 200
```

store

store_from_data_file

```
supplier = Acme
item = 10
vol = 200
```

OK? yes

Another way of performing the operation is:

```
linus: store supply -brief
```

```
supplier?      Acme
item?         10
vol?         200
```

*

Request: store_from_data_file, sdf

This request reads data from a file and loads it into the specified table. It may be used to reload data written by the write_data_file request.

Usage

```
sdf table_name -control_args
```

where:

1. table_name
is the name of the table defined in the open model or submodel.
2. control_args
can be chosen from the following:
 - column_delimiter CHAR, -cdm CHAR
where CHAR is a single ASCII character used to delimit the column values.
(Default column delimiter is the tilde.)
 - input_file pathname, -if pathname
specifies the name of the file which contains the input data.
 - row_delimiter CHAR, -rdm CHAR
where CHAR is a single ASCII character used to delimit the row values.
(Default row delimiter is the newline (NL) character.)

Note

One level of quotes is removed from each column value, if present.

Examples

```
store_from_data_file employee -if employee_data
store_from_data_file employee -if employee_data -cdm X -rdm Y
```

store_from_data_file

subsystem_name

Several rows could be added to the supply table by first creating the following file with a text editor:

```
Acme,10,200,  
XYZ,12,150,  
J. Smith,10,100,
```

and then entering the following request:

```
linus: store_from_data_file supply -input_file supply_file  
      -column_delimiter,
```

Request: string

This request returns a single character string formed by concatenating all of the strings together, separated by single spaces.

Usage

```
string {STRs}
```

Usage as an Active Request

```
[string {STRs}]
```

If no STRs are specified, a null character string is returned. If one or more STRs are specified, any quotes are returned as single quotes.

Examples

```
string He said, "Hi."  
He said, Hi.
```

```
string He said, ""Hi.""  
He said, "Hi."
```

```
string [string This is "food".]  
This is food.
```

Request: subsystem_name

This request displays the name of the subsystem. As an active request, it returns the name of the subsystem.

Usage

subsystem_name

Usage as an Active Request

[subsystem_name]

Request: subsystem_version

This request displays the version number of the subsystem. As an active request, it returns the version number of the subsystem.

Usage

subsystem_version

Usage as an Active Request

[subsystem_version]

Request: translate_query, tq

This request translates the current query and makes it available for linux data manipulation requests.

Usage

tq

Note

Refer to Section 1 for examples of query statements.

write

write

Request: write, w

This request specifies that the selected data is to be retrieved and written to a specified Multics file. The output file is a text file created by `vfile_` in the `stream_output` mode. If the file already exists, it may optionally be extended, although normally it would be truncated. A translated or translatable query must be available.

Usage

w outfile {-control_args}

where:

1. outfile
is the pathname of a Multics file into which the selected data is to be written. If the file does not currently exist, it is created. If the file currently exists, it is truncated unless `-extend` is also specified.
2. control_args
may be one or more of the following:
 - extend
specifies that if the outfile exists, it is to be added to, rather than truncated.
 - column_delimiter CHAR, -cdm CHAR
specifies that each selected value is to be delimited by the CHAR character in the outfile. If not present, each selected value is delimited by one blank.
 - row_delimiter CHAR, -rdm CHAR
specifies that each row is to be delimited by the CHAR character in the outfile. If not present, each row is delimited by a newline (NL) character.

Note

The output file is a text stream file created by `vfile_`. Each set of selected values is delimited by a newline character or the row delimiter character, if specified. The output file is suitable for processing by a text editor, as well as

write

write_data_file

other Multics facilities which process ASCII text files.

Example

Create a text file consisting of the name and salary of every employee. The query is:

```
linus: input_query
Query:
select name sal
from emp
```

The text file may then be created by:

```
linus: write_salary_file
```

Request: write_data_file, wdf

This request retrieves the selected data and places it in an output file, in a format suitable for input to the store_from_data_file request.

Usage

```
wdf pathname {-control_args}
```

where:

1. `pathname`
is the name of the file where the data is to be written.
2. `control_args`
can be chosen from the following:
 - column_delimiter CHAR, -cdm CHAR
where CHAR is a single ASCII character used to delimit the column values.
(Default column delimiter is the tilde.)
 - create_columns STR, -crc STR
STR specifies the column positions for new columns with null values. STR
is a blank-separated list of numbers. (See "Notes" below.)
 - extend
specifies that if the file already exists, it should be extended rather
than truncated.
 - row_delimiter CHAR, -rdm CHAR
where CHAR is a single ASCII character used to delimit the row values.
(Default row delimiter is the newline (NL) character.)
 - truncate
specifies that if the file already exists, it should be truncated.
(Default)

Notes

The `-create_columns` control argument aids in the restructuring of tables. The column positions specified are the positions in the output file where the null value is placed. To create two new columns as the third and fifth columns in the output file, the string `"-create_columns 3 5"` would be used. The null value is provided by placing two column delimiters together without any intervening characters, and the zero length character string is converted according to the data type of the column when the `store_from_data_file` request processes the input file.

Column values are examined to determine if they contain quotes, column delimiters, or rowdelimiters. If any of these are found, the column value is requoted before it is placed in the output file. The `store_from_data_file` request removes this layer of quotes when processing the file.

Examples

```
write_data_file employee_data
write_data_file employee_data -extend
write_data_file employee_data -create_columns 1 5
write_data_file employee_data -column_delimiter X
write_data_file employee_data
  abel~1~14555.01~36~m~s~ak~juneau
  abell~2~13000.01~55~f~m~az~phoenix
  .
  .
  baker~1~12000.10~71~m~s~il~springfield
```

SECTION 6

OBSOLETE LINUS CONTROL ARGUMENTS/REQUESTS

This section contains descriptions of the pre-MR10.2 linus command and associated requests that are obsoleted with the installation of MR10.2. The requests (invoke and lila) have been replaced by new requests, but will remain fully supported for an indefinite period.

The invoke request has been replaced by "exec_com" and the functions of lila are replaced by the requests:

```
apply
qedx
input_query      (lila new)
print_query      (lila print)
save_query       (lila save)
translate_query  (lila proc)
```

All of the above (new) requests are described in Section 4.

Name: linus

This command invokes linus to access an MRDS data base. It provides both retrieval and update operations. Data to be selected is specified via lila requests.

Usage

linus {macro_path} {-control_args}

where:

1. macro_path

is an optional argument that specifies the pathname of an ASCII segment from which linus is to take its initial instructions. Such a set of instructions is referred to as a macro. If path does not have a suffix of linus, then one is assumed. However, the suffix linus must be the last component of the name of the segment. If macro_path is provided, linus executes the requests contained in the specified segment and then waits for the user to type further requests. If the -request control argument is provided, linus executes the specified requests and then waits for the user to type further requests. If both macro_path and -request are omitted, linus waits for the user to type a request. A discussion of linus macros is provided later in this section. The usage of this argument is incompatible with usage of the -request control argument below. (Default -- linus waits for instruction from user input.)

2. control_args

can be chosen from the following:

-abbrev, -ab

enables abbreviation expansion and editing of request lines.

-arguments macro_args, -ag macro_args

where macro_args are one or more character strings to be substituted for special strings in the macro segment. This control argument may be specified only if macro_path is provided.

Note: This control argument must be the last control argument given. The others may be given in any order.

-iteration, -it

recognizes parentheses in the request line to indicate request line iteration.

-no_abbrev, -nab

disables abbreviation expansion and editing of request lines. (Default)

-no_iteration, -nit

parentheses in the request line are interpreted literally (i.e., they do not cause request line iteration. (Default)

-no_prompt, -npmt

turns off prompting of strings. This control argument can be overridden

later (see `set_mode` request). (Default is `prompt`.)

- `-no_start_up, -nsu`
specifies that the subsystem `start_up exec_com` is not to be executed.
- `-profile path, -pf path`
specifies the pathname of the profile used for abbreviation expansion. A profile suffix must be the last component to path; however, the suffix need not be supplied in the command line. This control argument implies `-abbrev`.
- `-request STR, -rq STR`
executes `STR` as a `linus` request line before entering the request loop. The usage of this control argument is incompatible with the usage of the `macro_path` argument. (Refer to `macro_path` description above.)
- `-set_lila_prompt_string STR, -slaps STR`
sets the prompting string used by `lila` to `STR`. If `STR` contains embedded blanks, it must be enclosed in quotes. (Default `lila` prompt is `"->"`.)
- `-set_linus_prompt_string STR, -slups STR, -prompt STR`
sets the prompting string used by `linus` to `STR`. If `STR` contains embedded blanks, it must be enclosed in quotes. (Default `linus` prompt is `"linus:"`.)
- `-start_up, -su`
specifies that the subsystem `start_up exec_com "start_up.lec"` is executed prior to entering the request loop. The `start_up` is searched for in the user home directory, project directory, and then `>site`. (Default)

Notes

While most users interact with `linus` through a terminal, this facility is designed to accept input through the `user_input` I/O switch and to transmit output through the `user_output` I/O switch. These switches can be controlled, via the `io_call` command, to interface with other devices/files in addition to the user's terminal. For convenience, the `linus` description assumes that the user's input/output device is a terminal.

By default, `linus` prompts the user whenever input is expected from the user input I/O switch (the string `"linus:"` is displayed if at `linus` request level, or the symbol `"->"` is displayed if within the `lila` editor). Refer to the description of the `set_mode` request for information on how to turn off prompting.

Multics program interrupt conditions are recognized and handled by `linus`. Thus, the user may interrupt any request and resume the `linus` session by invoking the Multics program interrupt command. After the program interrupt command, `linus` waits for the user to type further requests.

There is no data base creation facility within `linus`. Those users who wish to create their own data base should refer to Section 3 for information on the creation of an MRDS data base.

LINUS Requests

`invoke, i`
executes requests in a designated linus macro segment.

`lila`
invokes the lila editor which is used to build and process lila expressions to select data for manipulation by subsequent linus requests.

All other linus requests are listed and described in Section 4.

Request: `invoke, i`

This request specifies that requests contained in the designated macro segment are to be executed. Arguments may optionally be passed to the macro. This feature provides the capability to invoke a pre-defined series of linus requests.

Usage

`i path {macro_args}`

where:

1. `path`
is the pathname of the ASCII segment containing the linus macro. If `path` does not have a suffix of `linus`, then one is assumed. However, the suffix `linus` must be the last component of the name of `path` segment.
2. `macro_args`
are character strings to be substituted for special strings in the macro segment.

Note

Upon acceptance of the `invoke` request, the macro segment is read and executed, line-by-line. Argument substitution also takes place on a line-by-line basis, after the line is read and prior to its execution. After all lines in the macro segment are processed, `linus` waits for the user to type further requests on the terminal. The macro facility is described in detail later in this section. The `invoke` request is not compatible with request line iteration.

linus

linus

Example

Execute the requests contained in the segment `get_salary.linus`, passing the argument "John Smith".

```
linus: i get_salary "John Smith"
```

Request: lila

This request invokes the lila editor which is used to build and process lila expressions. This is a line editor which is very similar to a basic editor. A processed lila expression must be available when a `print`, `assign values`, `write`, `report`, `create_list`, `modify`, `delete`, or `define_temp_table` request is specified. (Refer to Section 1 for a description of the selection language and the syntax and semantics of lila.)

Usage

```
lila {-control_args}
```

where `control_args` can be chosen from the following:

`-build {start} {increment}`

invokes lila build mode, an automatic numbering mode, in the current lila text file. The value of `start` determines the first line number of the inserted text. The value of `increment`, when added to the previous line number, yields the next automatic line number. A value for `start` must be given if an `increment` is to be given. Both `start` and `increment` must be positive integers ranging from 1 to 9999. Build mode is exited by entering a line consisting only of a period (`.`). The default `increment` is 10 and the default `start` is the current last line number plus the `increment`.

`-new`

specifies that text from previous invocations of the lila editor are to be deleted. By default, previous text is made available for further editing and processing.

Lila Requests

The lila editing requests are:

- `.` identifies the lila editor of linus and the linus version number.
- `?` displays a list of available requests.

build {start} {increment}
invokes build mode in the current lila text file. The value of start designates the first automatic line number. The value of increment designates the offset used to generate succeeding automatic line numbers. (See -build control argument above.)

execute, e
passes the rest of the request line to the Multics command processor.

invoke path {macro_args}, **i path** {macro_args}
executes requests contained in the linus macro segment designated by path after passing any specified macro_args. This request functions in the same fashion as the linus invoke request.

line_number
deletes the text line specified by line_number.

line_number text_line
adds or replaces the line of text in the proper sequence as specified by line_number. The line_number may have a value ranging from 0 to 9999.

list, ls
displays all text lines in the current lila file.

list_requests, lr
displays a brief summary of available lila requests.

new
deletes all text from the current lila text file.

proc
processes the lila expression source text contained in the current lila file to produce a processed lila expression suitable for use by subsequent linus data manipulation requests.

quit, q
terminates the current lila session, and places the user at linus request level. The contents of the current lila file are retained for possible manipulation in a subsequent lila session within the current linus session.

save path, sv path
writes the contents of the current lila file to the segment designated by path for use by a subsequent invoke. If not present, a suffix of linus is added to path.

Example

Build and process a lila expression to find all items sold by departments located on the second floor. In the following example, one line (30) is purposely input with a mistake which is subsequently corrected.

```
linus: lila -new
-> 10 select item
-> 20 from sales
-> 30 where dept = {select deppt
-> 40                from loc
-> 50                where floor = 2}
```

linus

linus

```
-> ls
0010 select item
0020 from sales
0030 where dept = {select deppt
0040                  from loc
0050                  where floor = 2}
```

The next entry corrects a typo "deppt" shown in line 30.

```
-> 30 where dept = {select dept
-> ls
0010 select item
0020 from sales
0030 where dept = {select dept
0040                  from loc
0050                  where floor = 2}
-> proc
-> q
```

MACRO FACILITY

Linus provides the capability to execute a series of requests contained in a text segment. Such a segment is referred to as a linus macro segment. The name of a linus macro segment must have a suffix of `linus`.

A linus macro may be invoked in one of two ways: 1) via the linus command line, or 2) via the `linus` or `lila` invoke requests. Invocation via the linus command line is:

```
linus macro_path -arguments arg1 ... argn
```

which is equivalent to the sequence:

```
linus
invoke macro_path arg1 ... argn
```

A linus macro segment contains a series of linus requests in the same format as if they were entered at the terminal. Comments may appear in a linus macro segment as they would in a PL/I source segment, with the exception that a comment must be contained within one line. Arguments to the linus macro can be specified in a method analogous to the specification of arguments to a Multics `exec com`. In a linus macro, strings of the form `%i%` are interpreted as dummy arguments and are replaced by the corresponding macro args in the invoke request or in the linus command line. For example, macro `arg1` is substituted for the string `%1%` and macro `arg10` is substituted for the string `%10%`. Substitutions are also made within quoted strings. If a `%` is to be included in a string, `%%` must be specified.

The following is an example of a linus macro that displays the sales volume, given a department name and item code:

```
o dept_store retrieval /* o data base */
ss sales r n /* allow read only, no prevents */
lila -new          /* specify the data */
10 select vol
20 from sales
30 where dept = "%1%" & item = %2%
proc
q
pr -no_header     /* no need for header */
ds *              /* clean up */
c
q
```

Assume this macro resides in the segment `volume.linus`. Then, in order to obtain the sales volume for item 20 in the shoe department, the user types:

```
linus volume -arguments shoe 20
```

and the resulting where clause reads:

```
where dept = "shoe" & item = 20
```

SECTION 7

EXEC COM FACILITY

The capability to execute a series of requests contained in a text segment is provided by `linus`. Such a segment is referred to as a `linus exec_com`. The name of a `linus exec_com` must have a suffix of `.lec`. A `linus exec_com` is executed by the sequence:

```
linus
linus: ec exec_com_path {arg1 ... argn}
```

A `linus` macro segment contains a series of `linus` requests in the same format as they were entered at the terminal. It is possible to specify arguments to the `linus exec_com`. In the `linus exec_com`, strings of the form `&1` are substitutable arguments and are replaced by the corresponding `exec_com` arguments in the `exec_com` request line. For example, `exec_com` argument 1 is substituted for the string `&1`, and `exec_com` argument 10 is substituted for the string `&10`.

An example of a `linus exec_com` that displays the sales volume, given a department name and item code, is:

```
&version 2
&trace off                &- no need to see requests as they execute
&attach                    &- have linus read lines from here
&if &[e equal &n 2]        &- make sure two args were supplied
&then                      &- yes they were
&else &do                  &- no they weren't, print usage and return
    &print Usage: "ec volume dept item"
    &return
&end
&if &[open dept_store r]   &- open data base
&then &if &[set_scope sales r n] &- allow read only, no prevents
    &then &goto continue   &- scope was set
    &else close             &- scope wasn't set
&print The data base is not available. Try again later.
&return
&label continue
input_query -brief -force  &- specify the data
select vol
from sales
where dept = "&1" && item = &2 &- must specify 2 &&s to get 1 &
.
sfo -tl off -pl 0          &- turn title line off and set page length to 0
display                    &- display the data
del_scope *                &- delete all scope
close                      &- close the data base
&detach                    &- have linus read lines from terminal again
&quit                      &- and return to linus
```

Assume this `exec_com` resides in the segment `volume.lec`. Then, in order to obtain the sales volume for item 20 in the shoe department, the user enters:

linus
linus: ec volume shoe 20

and the resulting where clause reads:

where dept = "shoe" & item = 20

APPENDIX

STATIC DATA PARAMETERS

The following parameters were used during the generation of the LINUS system software.

Default buffer size - 256 words (linus_data_\$buff_len).
Default value for the maximum number of arguments a scalar function may take - 20 (linus_data_\$max_self_items).
Maximum depth of invoke nesting - 20 (linus_data_\$max_invocs).
Maximum length of a linus request - 5000 characters (linus_data_\$req_buf_len).
Maximum number of arguments to linus - 100 (linus_data_\$max_req_args).
Maximum number of items in a from clause - 20 (linus_data_\$max_range_items).
Maximum number of items in a select clause - 100 (linus_data_\$max_user_items).
Maximum number of LINUS variables (using the assign_value request) - 20 (linus_data_\$max_lvars).
Maximum number of MRDS items not previously selected that may occur in an expression - 20 (linus_data_\$max_expr_items).
Maximum number of set operators that may be stacked - 10 (linus_data_\$max_set_stack_size).
Maximum number of temporary tables - 20 (mrds_data_\$max_temp_rels).
Maximum string size - 500000 bits (linus_data_\$lit_string_size).
Number of spaces between columns for the print request - 2 (linus_data_\$print_col_spaces).
Print buffer length - 5000 characters (linus_data_\$pr_buff_len).

The following parameters were used for the generation of linus_lila_tokens_, which are used to define the keywords of a select statement. This segment is not bound in with linus.

```
select - linus_lila_tokens_$select
from   - linus_lila_tokens_$from
where  - linus_lila_tokens_$where
inter  - linus_lila_tokens_$inter
union  - linus_lila_tokens_$union
differ - linus_lila_tokens_$differ
unique - linus_lila_tokens_$unique
dup    - linus_lila_tokens_$dup
```

**MULTICS LOGICAL INQUIRY AND
UPDATE SYSTEM REFERENCE MANUAL
ADDENDUM A**

SUBJECT

Changes to the Manual

SPECIAL INSTRUCTIONS

This is the first addendum to AZ49-03, dated December 1983. Insert the attached pages into the manual according to the collating instructions on the back of this sheet.

Throughout the manual, change bars in the margins indicate technical additions and asterisks denote deletions. Section 7 is new and does not contain change bars.

Note:

Insert this sheet behind the manual cover to indicate the updating of the document with Addendum A.

SOFTWARE SUPPORTED

Multics Software Release 12.0

ORDER NUMBER

AZ49-03A

August 1986

COLLATING INSTRUCTIONS

To update the manual, remove old pages and insert new pages as follows:

Remove

TP, Preface

iii, iv
v, blank

1-5 through 1-10

2-7, 2-8

4-15 through 4-18

4-23, 4-24

4-33 through 4-36

4-43 through 4-46

4-53 through 4-62

5-1 through 5-74

A-1, blank

i-1 through i-4

TP Remarks Form (12-83)

Insert

TP, Preface

iii through vi

1-5 through 1-10

2-7, 2-8

4-15 through 4-18

4-23, 4-24

4-33 through 4-36

4-43 through 4-46

4-53 through 4-62

5-1 through 5-72

5-73, blank

7-1, 7-2

A-1, blank

i-1 through i-4

TP Remarks Form (8-86)

The information and specifications in this document are subject to change without notice. Consult your Honeywell Marketing Representative for product or service availability.

INDEX

- ! see exclamation mark
- " see quotes
- > see prompting mode
- see dot
- ; see semicolon 4-2,
- ? see prompting mode
- ^ see prompting mode
- abbreviations
 - dup (duplicate)
 - LILA (LINUS Language)
 - LINUS (Logical Inquiry and Update System)
 - linus command
 - requests
 - ab (abbrev)
 - ap (apply)
 - av (assign_values)
 - c (close)
 - cls (create_list)
 - clv (column_value)
 - d (delete)
 - dcl (declare)
 - di (display)
 - dib (display_builtins)
 - dltt (delete_temp_table)
 - ds (del_scope)
 - dtf (define_temp_table)
 - e (execute)
 - ec (exec_com)
 - fl (format_line)
 - h (help)
 - i (invoke)
 - iq (input_query)
 - ldb (list_db)
 - lh (list_help)
 - lr (list_requests)
 - ls (list_scope)
 - lsfo (list_format_options)
 - lv (list_values)
 - m (modify)
 - o (open)
 - odb (opened_database)
 - p (print)
 - pic (picture)
 - pq (print_query)
 - q (quit)
 - q (see lila request)
 - qx (qedx)
 - rpt (report)
 - rsfo (restore_format_options)
- abbreviations (cont.)
 - s (store)
 - sfo (set_format_options)
 - ss (set_scope)
 - svfo (save_format_options)
 - tq (translate_query)
 - w (write)
 - MRDS (Multics Relational Data Store)
 - MRPG (Multics Report Program Generator)
 - slaps (set_lila_prompt_string)
 - slups (set_linus_prompt_string)
- abs
 - see functions
- after
 - see functions
- arithmetic expression 1-4
- assign entry
 - see functions
- asterisk 1-3
- avg
 - see functions
- Backus-Naur Form 1-7
- before
 - see functions
- braces 1-4
- built-in functions
 - see functions
- calc entry
 - see functions
- ceil
 - see functions
- character string constants
 - see constants
- clause
 - from 1-2
 - select 1-2, 1-3
 - asterisk 1-3
 - restriction 5-18, 5-20, 5-44
 - where 1-2
 - if omitted 1-9
 - logical operators 1-3
 - parentheses 1-3
 - relational operators 1-3
 - where" 1-3
- column 3-1
 - names
 - see names

column (cont.)
 values
 specify 5-44

command processor
 invoking 5-21

concat
 see functions

constants
 character string 1-3
 quotes 1-3

count
 see functions

data base
 access 5-2, 6-2
 add to 5-66
 example 5-67
 closing 5-14
 create lister file 5-15
 creation 3-1, 5-3, 6-3
 example 3-1
 delete rows 5-20
 display 5-48
 display open 5-35
 example 1-2, 3-1
 modify 5-44
 open 5-46
 update example 5-65

data submodel 5-46

designators
 row 1-6, 1-10

differ (set operation)
 see operations

display data 5-48

domain 1-3, 1-9, 3-1

dot 5-4, 5-8

duplicate (dup)
 see selected values

evaluation
 order of 1-10
 union, differ, and inter 1-9
 use of braces 1-9
 use of parentheses 1-10

exclamation mark 5-12

exec com
 facility 7-1

file
 Multics
 generation 5-71
 output 5-71
 generation 5-71

floor
 see functions

formatted report 5-15, 5-51

from clause
 see clause

functions
 built-in 1-9, 2-1
 arithmetic scalar
 abs 2-1
 ceil 2-2
 floor 2-3
 mod 2-5
 round 2-5
 arithmetic set
 avg 2-2
 count 2-3
 max 2-4
 min 2-4
 sum 2-6
 character string scalar
 search 2-6
 verify 2-7
 examples
 arithmetic set -- avg 1-4, 1-5,
 5-16, 5-18
 arithmetic set -- max 1-4
 string scalar -- substr 5-16
 string scalar
 after 2-1
 before 2-2
 concat 2-3
 index 2-4
 reverse 2-5
 substr 2-6
 declared 5-17
 installation-defined 2-7
 nonstandard 2-7, 5-16
 restrictions 2-7
 scalar 2-7, 5-16, 5-17
 set 1-4, 2-7, 5-16
 assign entry 2-7
 calc entry 2-7
 sets 5-16

identifier
 dollar sign 1-10
 hyphen 1-10

index
 see functions

installation-defined functions
 see functions

inter (set operation)
 see operations

key column 1-2, 5-18, 5-44

LILA
 also see abbreviations
 expression
 example 6-6

lila requests
 summary of 6-5

line editor
 lila 6-5

LINUS
 also see abbreviations
 session
 terminate 5-51
 variables
 using set 5-12

linus command 5-2, 6-2
 requests
 : 5-8
 ? 5-8

linus command (cont.)

- abbrev 5-8
- answer 5-9
- apply 5-11
- assign values 5-12
- close 5-14
- column value 5-14
- create_list 5-15
- declare 5-16
- define temp table 5-17
- del_scope 5-19
- delete 5-20
- delete temp table 5-21
- display 5-21
- display builtins 5-25
- do 5-26
- exec com 5-28
- execute 5-29
- format_line 5-30
- help 5-32
- if 5-33
- input_query 5-34
- invoke 6-4
- lila 6-5
- list_db 5-35
- list_format_options 5-37
- list_help 5-40
- list_requests 5-41
- list_scope 5-42
- list_values 5-42
- ltrim 5-43
- modify 5-44
- open 5-46
- opened database 5-47
- picture 5-48
- print 5-48
- print_query 5-50
- qedx 5-50
- quit 5-51
- report 5-51
- restore_format_options 5-52
- rtrim 5-53
- save_format_options 5-53
- save_query 5-55
- set_format_options 5-55
- set_mode 5-63
- set_scope 5-64
- store 5-66
- store from data_file 5-68
- string 5-69
- subsystem_name 5-69
- subsystem_version 5-70
- translate_query 5-70
- write 5-71
- write_data_file 5-72
- set_lila_prompt_string 6-2
- set_linus_prompt_string 5-2, 6-3
- summary of requests 5-4, 6-3

lister command 5-15

- formatted report 5-15

lister file 5-15

- create 5-16

logical operators

- see operators

macro 6-2, 6-4

- facility 6-8
- invoke 6-8
- example 6-8
- segment 6-8
- execute 6-4

max

- see functions

metalanguage

- symbols 1-7
- underscore of 1-7

min

- see functions

mod

- see functions

mode

- set or reset 5-63

MRDS

- see abbreviations

MRPG

- see abbreviations

names

- column 1-3, 1-6
- table 1-6
- variable 5-12
- exclamation mark 5-12

nesting 1-6

nonstandard functions

- see functions

null strings 5-45

operations

- set
 - differ 1-5
 - inter 1-5
 - union 1-5
 - union, inter, and differ 1-9
 - union-compatible 1-9

operators

- arithmetic 1-3
- evaluation precedence 1-10
- logical 1-3
- relational 1-3

order of evaluation

- see evaluation

output file 5-71

parameters

- static data A-1

parentheses 1-3

program_interrupt 5-3, 6-3

prompting mode 5-63

- > 5-3, 5-63, 6-3
- also see set_lila_prompt_string
- ? 5-3, 6-3
- also see set_linus_prompt_string
- ^ 5-63
- linus: 5-63

question mark

- see prompting mode

- quotes 1-3
- relational operators
 - see operators
- report generation 5-51
- report writer 4-1
 - default report elements 4-5
 - format options 4-2
 - active 4-3
 - general column 4-2
 - general report 4-2
 - requests 4-4
 - specific column 4-3
 - values 4-3
 - full page formatting 4-10
 - optional report elements 4-6
 - user session 4-13
- requests
 - see linux command
- reverse
 - see functions
- round
 - see functions
- row 1-2, 3-1
 - designators
 - see designators
- scalar functions
 - see functions
- scope of access 5-19, 5-42
 - defining 5-64
 - operation codes 5-65
- search
 - see functions
- select clause
 - see clause
- select-from-where block 1-2
 - inner
 - braces 1-4
 - set functions
 - see functions
- selected values
 - duplicate 1-9
 - unique 1-9
- selection language 1-1
 - examples 1-1
- semicolon 4-23
- set functions
 - see functions
- set operations
 - see operations
- set_lila_prompt_string 5-63, 6-2
- set_linus_prompt_string 5-63, 6-3
- substr
 - see functions
- sum
 - see functions
- symbols
 - see metalanguage
- Syntax and Semantics of the Selection Language 1-7
- table 1-1, 1-6, 3-1
 - names
 - see names
- temporary table 5-17
 - expression simplification 5-18
 - restriction 5-18, 5-36
- terminology differences (LINUS/MRDS)
 - 3-1
- text segment 6-8
- union (set operation)
 - see operations
- unique
 - see selected values
- user interaction 6-3
- variable 1-6
 - list 5-12
 - name
 - see names
- verify
 - see functions
- where clause
 - see clause

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE

MULTICS LOGICAL INQUIRY AND UPDATE
SYSTEM REFERENCE MANUAL
ADDENDUM A

ORDER NO.

AZ49-03A

DATED

AUGUST 1986

ERRORS IN PUBLICATION

[Empty box for errors in publication]

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

[Empty box for 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 _____

CUT ALONG LINE

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

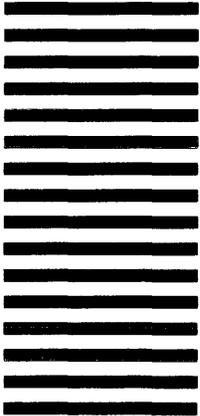


NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA 02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154



ATTN: PUBLICATIONS, MS486

CUT ALONG LINE

FOLD ALONG LINE

FOLD 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.

39384, 7.5C1283, Printed in U.S.A.

AZ49-03